IBM

PowerPC 401
Evaluation Board Kit
User's Manual

92G8619 000003

Fourth Edition (March 1998)

This edition of the *IBM PowerPC 401 Evaluation Kit User's Manual* applies to the IBM PowerPC 401 Evaluation Board Kit and to all subsequent versions of the 401 Evaluation Board Kit until otherwise indicated in new versions or technical newsletters.

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS MANUAL "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

IBM does not warrant that the contents of this publication or the accompanying source code examples, whether individually or as one or more groups, will meet your requirements or that the publication or the accompanying source code examples are error-free.

This publication could contain technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or program(s) described in this publication at any time.

It is possible that this publication may contain references to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country. Any reference to an IBM licensed program in this publication is not intended to state or imply that you can use only IBM's licensed program. You can use any functionally equivalent program instead.

No part of this publication may be reproduced or distributed in any form or by any means, or stored in a data base or retrieval system, without the written permission of IBM.

Requests for copies of this publication and for technical information about IBM products should be made to your IBM Authorized Dealer or your IBM Marketing Representative.

Address comments about this publication to:

IBM Corporation
Department YM5A
P.O. Box 12195
Research Triangle Park, NC 27709

email: ppc400pubs@vnet.ibm.com

IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 1996, 1998. All rights reserved.

Printed in the United States of America.

4 3 2 1

Notice to U.S. Government Users–Documentation Related to Restricted Rights –Use, duplication, or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corporation.

Patents and Trademarks

IBM may have patents or pending patent applications covering the subject matter in this publication. The furnishing of this publication does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594, United States of America.

The following terms are trademarks of IBM Corporation:

PowerPC 401GF
AIX
AIXwindows
IBM
OS Open
PowerPC
PowerPC Architecture
RISC System/6000
RISCWatch
RISCTrace

Other terms which are trademarks are the property of their respective owners.

Contents

About This Book xix
Who Should Use This Book xix
How This Book is Organized xx
Contacting the IBM Embedded Systems Solution Center xxii
Related Publications xxiii
Overview of the 401 EVB 1-1
Introducing the 401 EVB Hardware Components 1-1
401 Evaluation Board 1-1
Cables and Power Supply 1-1
Introducing the 401 EVB Software Support Package 1-2
ROM Monitor 1-2
RISCWatch Debugger 1-2
IBM High C/C++ Compiler 1-2
OS Open Real-Time Operating System 1-3
Dhrystone Benchmark Program 1-3
Application Tools 1-3
Host System Requirements 2-1
RS/6000 Host System Requirements 2-1
PC Host System Requirements 2-2
SUN Host System Requirements 2-3
Installing the EVB Software 3-1
RS/6000 Installation 3-1
EVB Software Support Package Installation - RS/6000 3-1
RISCWatch Debugger Installation - RS/6000 3-4
PC Installation 3-4
EVB Software Support Package Installation - PC 3-4
RISCWatch Debugger Installation - PC 3-7
Sun Installation 3-7
EVB Software Support Package Installation - Sun 3-7
RISCWatch Debugger Installation - Sun 3-10
Host Configuration 4-1
RS/6000 Host Configuration 4-1

Serial Port Setup - RS/6000 4-1
Ethernet Setup - RS/6000 4-5
ROM Monitor-Debugger Communication Setup - RS/6000 4-7
PC Host Configuration 4-7
Serial Port Setup 4-8
Ethernet Setup - PC 4-10
Windows 3.1 4-11
Ethernet Setup - Windows 95 4-12
Ethernet Setup - Windows NT 3.51 4-13
ROM Monitor-Debugger Communication Setup - PC 4-13
Sun Host Configuration 4-13
Serial Port Setup - SUN 4-14
Ethernet Setup - SUN 4-14
ROM Monitor-Debugger Communication Setup - SUN 4-15

401 EVB Connectors 5-1

Serial Port Connectors 5-3
Ethernet Connectors 5-4
Parallel Port Connector 5-5
RISCWatch JTAG Debugger Connectors 5-7
Expansion/Test Interface Connector 5-8
Squall Expansion Interface 5-11
Power Connector 5-14
PCI Related Connectors 5-15
Setting the EVB Jumpers 5-17
Connecting the 401 EVB Hardware 5-21
Serial Port Connection 5-21
Ethernet Connection 5-22
Power Supply Connection 5-25
Using a Terminal Emulator 5-25
RS/6000 Terminal Emulation 5-26
PC Terminal Emulation 5-26
SUN Terminal Emulation 5-27
Booting the PowerPC 401 on the EVB 5-28

401 EVB Hardware 6-1

401GF Overview 6-2
Peripheral Components 6-3
Mechanical Specifications 6-5
Displays 6-5
Switches 6-6
Power Supply 6-6
Initialization 6-7
Endian Programming 6-8

Utility Port 6-9
Ethernet Remote DMA Ports 6-10
401 EVB Memory Map 6-11
Squall Expansion Interface 6-14
Non-Critical Interrupts 6-15
Critical Interrupts 6-16
Network Address of the Ethernet Controller 6-16

401 EVB ROM Monitor 7-1

ROM Monitor Source Code 7-1
Communications Features 7-2
Bootp and tftp Configuration to support ROM Monitor Loads 7-2
RS/6000 bootp and tftp configuration 7-2
PC bootp and tftp configuration 7-4
Automatic startup for Windows 3.1 and Windows NT 3.51 7-5
Automatic startup for Windows 95 7-6
SUN bootp and tftp configuration 7-6
Accessing the ROM Monitor 7-8
ROM Monitor Operation 7-8
Monitor Selections and Submenus 7-9
Initial ROM Monitor Menu 7-9
Selecting Power-On Tests 7-11
Selecting Boot Devices 7-13
Changing IP Addresses 7-15
Using the Ping Test 7-17
Entering the Debugger 7-19
Disabling the Automatic Display 7-21
Displaying the Current Configuration 7-22
Saving the Current Configuration 7-23
Setting the Baud Rate for S1 Boots 7-24
S1 Boot 7-26
Exiting the Main Menu 7-28
ROM Monitor User Functions 7-30
Flash Update Utility 7-30

401 EVB Sample Applications 8-1

Overview 8-1
ROM Monitor Flash Image 8-2
Using the Software Samples 8-4
Building and Running the Dhrystone Benchmark 8-6
Building and Running the usr_samp Program 8-7
Building and Running the timesamp Program 8-8
Resolving Execution Problems 8-9

Using the Ping Test on the ROM Monitor to Verify Connectivity 8-9
bootp and tftp Servers (Daemons) for ROM Monitor loads 8-10
Using OS Open Functions 8-10
Application Libraries and Tools 9-1
OS Open Libraries 9-1
Using Libraries and Support Software 9-4
Serial Port Support Library 9-4
Boot Library(RAM) 9-4
Input/Output Support Library 9-4
PowerPC Low-Level Processor Access Support Library 9-5
ROM Boot Library 9-5
ROM Monitor Ethernet IP Interface Library 9-5
Real-time Clock Interface Support Library 9-5
Integrated Ethernet IP Interface 9-5
Software Timer Tick Support Library 9-6
Device Drivers Supplied with the 401 EVB 9-7
Asynchronous Device Driver 9-7
Device Driver Installation 9-7
Device Installation 9-8
Opening Asynchronous Communication Ports 9-9
Reading and Writing 9-9
I/O Control 9-10
Polled Asynchronous I/O 9-12
Ethernet Device Driver 9-13
Device Driver Installation 9-13
ROM Monitor Ethernet Device Driver 9-14
ROM Monitor Ethernet Installation and Initialization 9-14
Liquid Crystal Display Device Driver 9-14
LCD Device Driver Installation 9-14
LCD Device Installation 9-15
Opening LCD Device 9-15
Writing to LCD 9-15
I/O Control 9-15
Environment Bringup and Initialization 9-17
Board bootstrap 9-17
Environment Initialization 9-18
Tools 9-19
elf2rom 9-19
hbranch 9-21
eimgbld 9-23
401 EVB Function Reference 10-1

Attributes and Threads 10-1
Async Safe Functions 10-1
Cancel Safe Functions 10-2
Interrupt Handler Safe Functions 10-2
401 EVB Functions 10-2

Program Trace Calls A-1

Overview A-1
MSGDATA Structure A-1
Ptrace Definitions A-4
RD_ATTACH (30) A-5
Request data A-5
Response data A-5
RD_CONTINUE (7) A-6
Request data A-6
Response data A-6
RD_DETACH (31) A-7
Request data A-7
Response data A-7
RD_FILL (105) A-8
Request data A-8
Response data A-8
RD_KILL (8) A-9
Request data A-9
Response data A-9
RD_LDINFO (34) A-10
Request data A-10
Response data A-10
RD_LOAD (101) A-12
Request data A-12
Response data A-12
RD_LOGIN (103) A-13
Request data A-13
Response data A-13
RD_LOGOFF (104) A-14
Request data A-14
Response data A-14
RD_READ_D (2) A-15
Request data A-15
Response data A-15
RD_READ_DCR (110) A-16
Request data A-16
Response data A-16

RD_READ_GPR (11) A-17
Request data A-17
Response data A-17
RD_READ_GPR_MULT(71) A-18
Request data A-18
Response data A-18
RD_READ_I (1) A-19
Request data A-19
Response data A-19
RD_READ_I_MULT (71) A-20
Request data A-20
Response data A-20
RD_READ_SPR (115) A-21
Request data A-21
Response data A-21
RD_STATUS (114) A-22
Request data A-22
Response data A-22
RD_STOP_APPL (113) A-23
Request data A-23
Response data A-23
RD_WAIT (108) A-24
Request data A-24
Response data A-24
RD_WRITE_BLOCK (19) A-25
Request data A-25
Response data A-25
RD_WRITE_D (5) A-26
Request data A-26
Response data A-26
RD_WRITE_DCR (112) A-27
Request data A-27
Response data A-27
RD_WRITE_GPR (14) A-28
Request data A-28
Response data A-28
RD_WRITE_I (4) A-29
Request data A-29
Response data A-29
RD_WRITE_SPR (112) A-30
Request data A-30
Response data A-30
RL_LDINFO (181) A-31

Request data A-31
Response data A-31
RL_LOAD_REQ(180) A-32
Request data A-32
Response data A-32
ROM Monitor Load Format B-1
Overview B-1
Section Types B-1
First Section B-2
Text Section B-3
Data Section B-3
Symbol Section B-3
Boot Header B-3
FPGA Program Images C-1
Peripheral Interface Controller C-1
Interleaved DRAM Controller C-15
SRAM Controller and Bus Arbiter C-45
401 EVB Bill of Materials D-1
Index X-1

Figures

Figure 5-1. 401EVB Connectors and Jumpers 5-2
Figure 5-2. Nine-Pin Serial Port Connector..... 5-3
Figure 5-3. RJ-45 Ethernet Connector (Front View) 5-4
Figure 5-4. Parallel Port Connector (J5) 5-5
Figure 5-5. RISCWatch JTAG Header (Top View)..... 5-7
Figure 5-6. 100-Pin Squall Receptacle 5-11
Figure 5-7. Serial Port Connection..... 5-22
Figure 5-8. Wiring in a Crossover Cable..... 5-23
Figure 5-9. Point-to-Point 10BaseT Ethernet Connection..... 5-24
Figure 5-10. 10BaseT Ethernet Connection with Hub..... 5-24
Figure 5-11. Point-to-Point 10Base2 Ethernet Connection..... 5-25
Figure 6-1. PowerPC 401GF Block Diagram 6-2
Figure 7-1. ROM Monitor Address Map 7-9
Figure 9-1. elf2rom Output File 9-20
Figure 9-2. Detail of patch file placement..... 9-22
Figure 9-3. hbranch Output Image..... 9-22

Tables

Table 5-1. Serial Port Signal Assignments 5-3
Table 5-2. Ethernet Connector Description 5-4
Table 5-3. Ethernet Connector Descriptions 5-5
Table 5-4. Connector J5 — Parallel Port 5-5
Table 5-5. RISCWatch JTAG Interface Connections and Resistors 5-7
Table 5-6. Connector J15 — Processor Expansion 1 5-8
Table 5-7. Connector J16 — Processor Expansion 2 5-10
Table 5-8. Connector J4 — Squall Interface 5-11
Table 5-9. Connector J11 — Power 5-15
Table 5-10. Connector J39 — Accessory +3.3V 5-15
Table 5-11. Connector J40 — PCI Socket +3.3V 5-16
Table 5-12. Connector J41 — PCI Socket and Squall +12V 5-16
Table 5-13. Connector J42 — PCI Socket and Squall -12V 5-16
Table 5-14. Jumper Programming of SIMM Size 5-17
Table 5-15. Jumper J9 — DRAM Programming PD0 5-17
Table 5-16. Jumper J10 — Arbiter Select 5-17
Table 5-17. Jumper J12 — PCI SERR Control 5-18
Table 5-18. Jumper J20 — Coax / Twisted-pair Selection 5-18
Table 5-19. Jumper J21 — Twisted-pair Link Integrity Checking 5-18
Table 5-20. Jumper J22 — Twisted-pair Good-link LED 5-19
Table 5-21. Jumper J26 — Parallel Port PP Mode 5-19
Table 5-22. Jumper J27 — PCI IDSEL Control 5-19
Table 5-23. Jumper J28 — DRAM Programming PD1 5-19
Table 5-24. Jumper J29 — 25/33 MHz Bus Frequency Selection 5-20
Table 5-25. Jumper J30 — DRAM Programming PD2 5-20
Table 5-26. Jumper J32 — PCI INTA Control 5-20
Table 5-27. Jumper J43 — PCI Clock Source 5-21
Table 6-1. Status LED Display 6-5

Table 6-2. 401 EVB Switches 6-6
Table 6-3. Utility Port Bit Assignments 6-9
Table 6-4. Ethernet DMA PRQ Status 6-10
Table 6-5. Ethernet Remote DMA Port 6-10
Table 6-6. Squall Module Memory Map 6-14
Table 6-7. Non-critical Interrupts, Controller 1 6-15
Table 6-8. Non-critical Interrupts, Controller 2 6-15
Table 6-9. Critical Interrupts 6-16
Table 9-1. OS Open Libraries 9-1
Table 9-2. OS Open Libraries for the 401 EVB 9-4
Table 9-3. ioctl() Commands for Asynchronous Device Drivers 9-10
Table 9-4. ioctl() Commands for the LCD Device Driver 9-16
Table 10-1. Functions Specific to 401 EVB 10-2
Table A-1. RD_ATTACH Request Table A-5
Table A-2. RD_ATTACH Response Table A-5
Table A-3. RD_CONTINUE Request Table A-6
Table A-4. RD_CONTINUE Response Table A-6
Table A-5. RD_DETACH Request Table A-7
Table A-6. RD_DETACH Response Table A-7
Table A-7. RD_FILL Request Table A-8
Table A-8. RD_FILL Response Table A-8
Table A-9. RD_KILL Request Table A-9
Table A-10. RD_KILL Response Table A-9
Table A-11. RD_LDINFO Request Table A-10
Table A-12. RD_LDINFO Response Table A-10
Table A-13. RD_LOAD Request Table A-12
Table A-14. RD_LOAD Response Table A-12
Table A-15. RD_LOGIN Request Table A-13
Table A-16. RD_LOGIN Response Table A-13
Table A-17. RD_LOGOFF Request Table A-14

Table A-18. RD_LOGOFF Response Table A-14
Table A-19. RD_READ_D Request Table A-15
Table A-20. RD_READ_D Response Table..... A-15
Table A-21. RD_READ_GPR Request Table A-17
Table A-22. RD_READ_GPR Response Table A-17
Table A-23. RD_READ_GPR_MULT Request Table A-18
Table A-24. RD_READ_GPR_MULT Response Table..... A-18
Table A-25. RD_READ_I Request Table..... A-19
Table A-26. RD_READ_I Response Table A-19
Table A-27. RD_READ_I_MULT Request Table A-20
Table A-28. RD_READ_I_MULT Response Table A-20
Table A-29. RD_READ_SPR Request Table..... A-21
Table A-30. RD_READ_SPR Response Table..... A-21
Table A-31. RD_STATUS Request Table..... A-22
Table A-32. RD_STATUS Response Table..... A-22
Table A-33. RD_STOP_APPL Request Table..... A-23
Table A-34. RD_STOP_APPL Response Table A-23
Table A-35. RD_WAIT Request Table..... A-24
Table A-36. RD_WAIT Response Table A-24
Table A-37. RD_WRITE_BLOCK Request Table A-25
Table A-38. RD_WRITE_BLOCK Response Table A-25
Table A-39. RD_WRITE_D Request Table..... A-26
Table A-40. RD_WRITE_D Response Table..... A-26
Table A-41. RD_WRITE_DCR Request Table..... A-27
Table A-42. RD_WRITE_DCR Response Table..... A-27
Table A-43. RD_WRITE_GPR Request Table..... A-28
Table A-44. RD_WRITE_GPR Response Table..... A-28
Table A-45. RD_WRITE_I Request Table A-29
Table A-46. RD_WRITE_I Response Table..... A-29
Table A-47. RD_WRITE_SPR Request Table..... A-30

Table A-48. RD_WRITE_SPR Response Table A-30
Table A-49. RL_LDINFO Request Table A-31
Table A-50. RL_LDINFO Response Table..... A-31
Table A-51. RL_LOAD_REQ Request Table A-32
Table A-52. RL_LOAD_REQ Response Table A-32
Table D-1. 401 EVB Bill of Materials D-2

About This Book

This book contains the information you need to install and use the IBM® PowerPC™ 401™ Evaluation Board (EVB), a hardware and software development tool for the PowerPC 401GF 32-bit RISC embedded controller.

Connection of the 401 EVB to a host system is required for the exercises in this book. Supported host systems include:

- an IBM RISC System/6000™ workstation running AIX™ 3.2.5 (or higher)
- an IBM or compatible PC running one of the following
    - Windows 3.1 (or higher) and a TCP/IP package compliant with the Microsoft Windows Socket API definition
    - Windows 95
    - Windows NT 3.51
- a Sun SPARCstation 5, 10, or 20 workstation running Solaris 2.3 (or higher) or SunOS 4.1.3 (or higher)

The 401 EVB hardware module comes with a 401GF controller, an Ethernet controller, 1MB flash memory, two serial ports, a parallel port, PCI host or adapter functionality, a two line LCD, 8 MB of DRAM, 512KB of SRAM, a real-time clock calendar with 8KB NV RAM, and expansion and test interfaces. The reference design also includes technical specifications and schematics.

The 401 EVB software includes the ROM Monitor (resident in the flash memory on the board), ROM Monitor source code, IBM's OS Open real time operating system, sample application programs, application development libraries and tools, IBM's High C/C++ compiler, and IBM's RISCWatch, a source-level debugger that runs on the host.

Who Should Use This Book

This book is for hardware and software developers who need to evaluate the 401GF embedded controller and use the debugging features of the 401 EVB to support software development.

Users should understand hardware and software development tools, concepts, and environments. Specifically, users should understand:

- the host's operating system
- the PowerPC Architecture™ and implementation-specific characteristics of the PowerPC 401GF embedded controller
- C and assembler language programming

How This Book is Organized

This book contains the following chapters and appendixes:

- Chapter 1, "Overview of the 401 EVB," describes the product, its hardware and software components, and its relationship with the software tools on the host.
- Chapter 2, "Host System Requirements," lists the hardware and software requirements of the host system.
- Chapter 3, "Installing the EVB Software," describes the software installation on the host system.
- Chapter 4, "Host Configuration," describes the steps required to facilitate communications between the host computer and the 401 EVB.
- Chapter 5, "401 EVB Connectors," describes the EVB connectors and the procedures for connecting and configuring the 401 EVB hardware.
- Chapter 6, "401 EVB Hardware," describes the hardware components and their functions in terms of the overall organization of the 401 EVB.
- Chapter 7, "401 EVB ROM Monitor," describes the operations of the ROM monitor.
- Chapter 8, "401 EVB Sample Applications," contains sample applications to be built, loaded onto the EVB, and run.
- Chapter 9, "Application Libraries and Tools," describes the application libraries and host tools provided with the EVB software.
- Chapter 10, "401 EVB Function Reference," lists the OS Open functions for the 401 EVB platform. The function calls are arranged alphabetically by function name.
- Appendix A, "Program Trace Calls," describes the messages for interfacing a debugger on the host system to the ROM monitor on the 401 EVB.
- Appendix B, "ROM Monitor Load Format," describes the load format requirements supported by the ROM monitor.
- Appendix C, "FPGA Program Images," contains the programming logic for the field progammable gate arrays on the 401 EVB.
- Appendix D, "401 EVB Bill of Materials," contains a list of parts used on the 401 EVB.

Conventions Used in This Book

This book follows the numeric and highlighting notation conventions based on those used in the RISC System/6000 and AIX publications.

Numeric Conventions

In general, numbers are used exactly as shown. Unless noted otherwise, all numbers are in decimal, and, if entered as part of a command, are entered without format information.

In text, binary numbers are preceded by a “B” followed by the number enclosed in single quotes, for example:

B'010'

In commands, binary numbers are preceded by “0b” or “b” followed by the number, which may be enclosed in single quotes, for example:

0b010 or b’010’

In text, hexadecimal numbers are preceded by an “X” followed by the number enclosed in single quotes, for example:

X'1A7'

In commands, hexadecimal numbers are preceded by “0x” or “x” followed by the number, which may be enclosed in single quotes, for example:

0x1a7 or x'1a7'

In text, the hexadecimal digits A through F appear in uppercase. In commands, these digits are typically entered in lowercase.

Highlighting Conventions

This book uses the following highlighting conventions:

- The names of invariant objects known to the software appear in bold type. In some text, however, such as in lists, no special typographic treatment is used. Examples of such objects include:
  - Function and macro names
  - Data types and structures
  - Constants and flags

Names of objects known to the software must be entered exactly as shown.

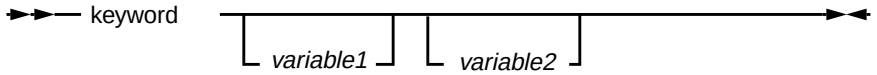
- Variable names supplied by user programs appear in italic type. In some text, however, such as in lists, no special typographic treatment is used. Examples of these objects include arguments and other parameters.
- No highlighting appears in code examples.

Syntax Diagram Conventions

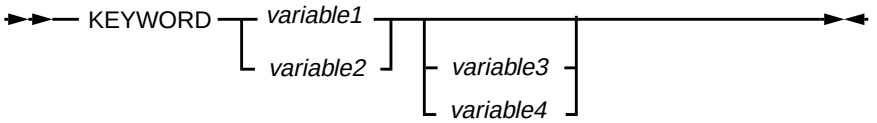
Throughout this book, diagrams illustrate the syntax for string formats and commands. The following list shows how to read these diagrams:

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.
- A ►►— symbol begins a diagram.
- A —► symbol indicates continuation of a diagram on the next line.
- A ►— symbol indicates continuation of a diagram from the previous line.
- A —►◄ symbol terminates a diagram.
- Keywords are in regular type, and variables are in italics. Keywords must be typed exactly as shown.
- Keywords or variables on the main path of a diagram are required.

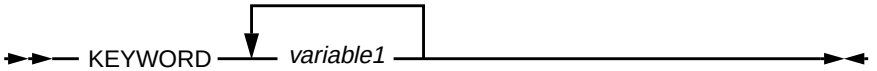
- Keywords or variables shown on branches below the main path are optional.



- Keywords or variables can appear in a stack, indicating that only one item in a stack can be chosen. If an item in a stack is on the main path, you must choose an item from the stack. If all items in a stack are below the main path, you may choose an item from the stack.
- For example, in the following syntax diagram, you must choose either *variable1* or *variable2*. However, because *variable3* and *variable4* are below the main path, neither is required.



- A repeat separator is a returning arrow that surrounds a syntax element or group and shows that the element or group can be repeated.



Contacting the IBM Embedded Systems Solution Center

For information about the 401 EVB Kit and the IBM family of hardware and software products for embedded system developers, call the IBM Embedded Systems Solution Center at (919) 254-1810.

Please send any comments regarding this document to the following Internet address:

ppc400pubs@vnet.ibm.com

Related Publications

Many of the following publications are included on the CD ROM that comes with the evaluation kit. The others are available from your IBM Microelectronics representative:

- **RISC System/6000 Publications**

  IBM RISC System/6000: POWERstation and POWERserver Hardware Technical Information General Architectures, SA23-2643

- **AIX Publications**

  This book refers to the following AIX publications. The words “IBM AIX Version 3.2 for RISC System/6000” are actually part of the title of each book; however, in all references to these books, those words are omitted.

  Assembler Language Reference, SC23-2197

  Commands Reference, Volume 1, SC23-2376

  Commands Reference, Volume 2, SC23-2366

  Commands Reference, Volume 3, SC23-2367

  Commands Reference, Volume 4, SC23-2393

  Editing Concepts and Procedures, GC23-2212

- **Embedded Application Binary Interface (EABI) Publications**

  PowerPC Embedded Application Binary Interface (EABI)

  System V Application Binary Interface, Third Edition, 0-13-0100439-5

  System V Application Binary Interface, PowerPC Processor Supplement

- **IBM High C/C++ Publications**

  The following list includes the books in the IBM High C/C++ library:

  IBM High C/C++ Programmer’s Guide for PowerPC, 92G6920

  IBM High C/C++ Language Reference for PowerPC, 92G6923

  IBM ELF Assembler User’s Guide for PowerPC, 92G6921

  IBM ELF Linker User’s Guide for PowerPC, 92G6922

- **OS Open Publications**

  The following list includes the books in the OS Open library:

  IBM OS Open Programmer's Reference, Volume 1, 92G6911

  IBM OS Open Programmer's Reference, Volume 2, 92G6912

  IBM OS Open User's Guide, 92G6897

- **RISCWatch Debugger Publications**

  RISCWatch Debugger User's Guide, 13H6964

- **PowerPC 400Series User's Manuals**

  PPC403GA Embedded Controller User's Manual, 13H6960

  PPC403GB Embedded Controller User's Manual, 13H6985

  PPC403GC Embedded Controller User's Manual, 13H6986

  PPC403GCX Embedded Controller User's Manual

  PPC401GF Embedded Controller User's Manual, 13H6948

1

Overview of the 401 EVB

This chapter introduces the hardware and software in the 401 EVB kit.

1.1 Introducing the 401 EVB Hardware Components

The 401 EVB kit contains the evaluation board with its power supply, line cord, serial port and Ethernet cables.

1.1.1 401 Evaluation Board

The 401 EVB is a full featured prototyping board which comes with the PowerPC 401GF embedded controller, 1MB of flash memory (the upper 512KB comes preprogrammed with the ROM Monitor), 8MB of DRAM, 512KB of SRAM, two serial ports, two 82C59 interrupt controllers (cascaded), a parallel port, an Ethernet controller, a real-time clock, 8KB NV RAM, PCI host/adapter functionality, and an expansion interface connector. Two DRAM slots are provided to support up to 64MB when both slots are populated with 32MB SIMMs.

The dual serial ports and parallel port connect to a National NS16C553 serial communications controller. The Ethernet controller is a National DP83902. The real-time clock/calendar is supplied by a Dallas Semiconductor DS1643L-12. The PCI bus bridge chip is a PLX Technology PCI9060-3. Four 128Kx8 modules of 12 nsec SRAM in 32-pin 400-mil SOJ socketed packages provide the 512KB of SRAM.

Product documentation for devices other than the 401 can be obtained from the respective manufacturers. Configuration and addressing information for all these devices is included in the subsequent chapter on the 401 EVB hardware.

Header connectors are provided for optional test equipment such as the RISCWatch™ JTAG debugger. This tool allows non-intrusive hardware and software debug through the 401 EVB JTAG port. For more information on the RISCWatch JTAG tool, call the IBM Embedded Systems Solution Center at (919) 254-1810.

1.1.2 Cables and Power Supply

The 401 EVB kit includes a serial port interface cable for connecting the EVB serial port 1 to a terminal or terminal emulator running on the host.
Note: The Sun version of the EVB kit contains a male-to-male adapter to support connectivity between serial port 1 on the EVB and a serial port on the host.

An Ethernet crossover cable is provided in the kit to support direct Ethernet communication with the host system. Standard 10Base2 and 10BaseT Ethernet connectors are provided on the EVB. The Ethernet crossover cable is for direct connection to a single host and cannot be used with a hub or a building's Ethernet network.

A power supply with line cord is also provided with the 401 EVB kit.

1.2 Introducing the 401 EVB Software Support Package

The 401 EVB software support package consists of the ROM Monitor, ROM Monitor source code, the RISCWatch source level debugger for ROM Monitor and OS Open debug modes, the IBM OS Open real time operating system, several sample programs (including the Dhrystone benchmark program), and application development libraries and tools. The IBM High C/C++ compiler is also included.

1.2.1 ROM Monitor

The ROM Monitor program for the 401 EVB is supplied in one of the 512KB flash memory modules on the board. This code initializes the 401 processor and the controllers for serial and Ethernet communications. By supporting communications with the host computer system, the ROM Monitor provides the means to load applications from the host onto the EVB and to debug them with the RISCWatch source level debugger.

The ROM Monitor is accessed through a terminal (or terminal emulator) attached to serial port 1 on the EVB. The RISCWatch debugger, when in ROM Monitor mode, runs on the host system, communicating with the ROM Monitor through serial port 2 or the Ethernet interface on the 401 EVB.

The ROM Monitor source code is provided primarily for customers interested in developing their own ROM versions. It is also provided so that debuggers other than RISCWatch may be integrated with the 401 EVB. Appendix A describes the trace calls that support communication between the RISCWatch debugger on the host and the ROM Monitor running on the 401 EVB.

1.2.2 RISCWatch Debugger

The RISCWatch source level debugger provides a window-based debugging environment for application programs running on the 401 EVB. The debugger can be used to load and execute application programs on the evaluation board. Debugger installation and usage for ROM Monitor and OS Open (non-JTAG) targets are addressed in the *RISCWatch Debugger User's Guide* included in the EVB kit. A sample debug session is included with the debugger.

1.2.3 IBM High C/C++ Compiler

The IBM High C/C++ compiler is a globally optimizing compiler developed for the PowerPC

family of processors. It produces executable code in Extended Link Format(ELF) file format. The version included in the software support package is a limited capacity version created specifically for the 401 EVB kit. It supports the compilation, assembly, and linkage of the sample application programs and the ROM Monitor source code. A full featured version of the IBM High C/C++ compiler is available from IBM. For more information call the PowerPC Embedded Systems Solutions Center at (919)254-1810.

1.2.4 OS Open Real-Time Operating System

OS Open is a real-time operating system (RTOS) available for the PowerPC 400 Series, 60x and 7xx processors. OS Open is designed to take full advantage of the power of the IBM PowerPC RISC processors. Also, because the OS Open environment is built in a scalable fashion, it can be configured to meet the functional requirements and memory constraints of a wide variety of embedded systems.

OS Open features:

- Hard real-time support, including deterministic execution, priority inheritance protocols, and priority ceiling protocols
- Board support packages for plug-and-play operation of popular board-level products
- Support for existing American National Standards Institute (ANSI) C and emerging POSIX standards
- Open network interfaces to support embedded systems in heterogeneous environments
- Scalable implementations to meet the requirements and constraints of a variety of embedded systems

The version of OS Open included in the EVB software contains a limited function kernel that limits the number of threads that can be in existence at one time. Additional details can be found in the readme file following software installation. A full function OS Open kernel is available from IBM. Contact the IBM Embedded Systems Solutions Center at (919)254-1810 for additional information.

1.2.5 Dhrystone Benchmark Program

The Dhrystone benchmark is a commonly available integer benchmark. It is included as an example program to be built, loaded onto the evaluation board, and executed. The results of this benchmark may vary based on compiler options and the system environment in which it is run.

1.2.6 Application Tools

Several host-based tools are provided to support ROM and application development on the 401 EVB.

2

Host System Requirements

This chapter describes the hardware and software requirements of the host system to which the 401 EVB is to be connected. Supported host systems include:

- an IBM RS/6000 workstation running AIX 3.2.5 (or higher)
- an IBM or compatible PC running one of the following
    - Windows 3.1 (or higher) and a TCP/IP package compliant with the Microsoft Windows Socket API definition
    - Windows 95
    - Windows NT 3.51
- a Sun SPARCstation 5, 10, or 20 workstation running Solaris 2.3 (or higher) or SunOS 4.1.3 (or higher)

2.1 RS/6000 Host System Requirements

Hardware requirements of the host RS/6000 computer include:

- Approximately 25MB of free disk space. This space is required for the IBM High C/C++ compiler, the 401 EVB Software Support Package, and the RISCWatch debugger. When planning disk space usage, consider disk space requirements for AIX and any other software packages.
- Two available serial ports, one for terminal emulation and the other for host-to-EVB communications. Only one serial port is required if an Ethernet adapter is available for host-to-EVB communications. For better performance, an Ethernet connection is strongly recommended. Most RS/6000 computers come equipped with two serial ports and an Ethernet adapter. Please consult your RS/6000 literature for more details.
- A graphics display (IBM 6091 or similar), to display debugger screens

The following software must be installed on the host RS/6000 computer to run the debugger that communicates with the ROM Monitor on the 401 EVB:

- RISCWatch 3.3 or higher
- AIX Version 3.2.5 or higher
- AIX/Windows™ with X11R5 and Motif 1.2

AIX tools used to develop OS Open applications include:

ELF

- High C/C++ compiler for C programs
- asppc assembler for assembler and C language programs
- eimgbld, binary image build tool
- ELF linker/binder, to build OS Open applications for a target system

IBM and other vendors provide numerous optional software development tools for AIX, including tools for:

- Computer-aided software engineering (CASE)
- Structured analysis and design
- Program understanding
- Code management and version control

2.2 PC Host System Requirements

Hardware requirements of the host PC include:

- IBM or compatible system unit. Minimum requirements: x486 DX2 50/66 MHz with 8 MB of RAM
- VGA/SVGA Display Monitor. Minimum required: VGA 640x480. Recommended: SVGA 1024x768
- Approximately 25MB of free disk space. This space is required for the IBM High C/C++ compiler, the 401 EVB Software Support Package, and the RISCWatch debugger. When planning disk space usage, consider disk space requirements for Windows and any other software packages.
- Two available serial ports, one for terminal emulation and the other for SLIP host-to-EVB communications. Since PC hardware varies greatly, you should consult your PC literature to determine the number of serial ports available. Only one serial port is required if an Ethernet adapter is available for host-to-EVB communications. For better performance, an Ethernet connection is strongly recommended. Establishing an Ethernet host-to-EVB connection will most likely require the installation of an Ethernet adapter card and some additional connectivity hardware since most PCs do not come equipped for Ethernet communications. That hardware might include any or all of the following:
    - For 10BaseT, an Ethernet 10BaseT network transceiver, a twisted pair cable, and a hub. At a minimum, a 10BaseT point-to-point connection will require the Ethernet crossover cable supplied with the EVB kit.
    - For 10Base2, an Ethernet/IEEE 802.3 10Base2 network transceiver, two BNC “T” type connectors, two terminating resistors, and a thin coaxial cable. At a minimum, a 10Base2 point-to-point connection will require one thin coaxial cable, two BNC “T” connectors, and two BNC terminating resistors.

The following software must be installed on the host PC to run the debugger that communicates with the ROM Monitor on the 401 EVB:

- RISCWatch 3.3 or higher
- Windows 3.1 or higher, Windows 95, or Windows NT 3.51

Windows 3.1 users require a TCP/IP package compliant with the Microsoft Windows Socket API definition. One such compatible TCP/IP package is Trumpet Winsock, a TCP/IP protocol stack available from the **www.trumpet.com** Internet site. Windows 95 users who want to establish a SLIP host-to-EVB connection over a second serial port, require Trumpet Winsock as well, since the TCP/IP package that comes with Windows 95 does not support SLIP communications. Appropriate installation documentation can be found at the Trumpet site. Users should refer to the documentation for the terms and conditions of using Trumpet Winsock. Information regarding the setup and use of Trumpet Winsock can be found in the subsequent chapter on “Host Configuration”.

Note: Trumpet is not recommended for Windows 95 users already connected to a network since installing Trumpet may cause problems with previously defined networks. If the recommended Ethernet host-to-EVB connection is going to be used (instead of the SLIP host-to-EVB connection), Windows 95 users do **not** need to install Trumpet since the TCP/IP package that comes with Windows 95 can be used to establish the Ethernet connection.

2.3 SUN Host System Requirements

Hardware requirements of the host Sun workstation include:

- Approximately 25MB of free disk space. This space is required for the IBM High C/C++ compiler, the 401 EVB Software Support Package, and the RISCWatch debugger. When planning disk space usage, consider disk space requirements for the operating system and any other software packages.
- An available serial port for terminal emulation and an Ethernet (Attachment Unit Interface (AUI) or RJ-45) port for host-to-EVB communications. Most Sun SPARCstations come equipped with one serial port and an Ethernet (AUI) port. Consult your Sun literature for additional details.
- Any or all of the following hardware to establish an Ethernet connection between the EVB and the host.
    - For 10BaseT, an Ethernet 10BaseT network transceiver, a twisted pair cable, and a hub. At a minimum, a 10BaseT point-to-point connection will require the Ethernet crossover cable supplied with the EVB kit.

    - For 10Base2, an AUI (or thick Ethernet) adapter cable (or an AUI/Audio Adapter cable depending on your SPARCstation model and options - both are available from Sun), an Ethernet/IEEE 802.3 10Base2 network transceiver, two BNC “T” type connectors, two terminating resistors, and a thin coaxial cable. At a minimum, a 10Base2 point-to-point connection will require one thin coaxial cable, two BNC “T” connectors, and two BNC terminating resistors.
    - Consult your hardware documentation for additional information.
- A graphics display to display debugger screens

The following software must be installed on the Sun workstation to run the debugger that communicates with the ROM Monitor on the EVB:

- RISCWatch 3.3 or higher
- SunOS 4.1.3 (or higher) or Solaris 2.3 (or higher)
- OpenWindows 3.0 (SunOS 4.1.3) or 3.3 (Solaris 2.3)

3

Installing the EVB Software

This chapter describes the procedures for installing the EVB software on the host system. Details of the software, its directories and their contents, are also given. Please refer to the section corresponding to your host system.

3.1 RS/6000 Installation

3.1.1 EVB Software Support Package Installation - RS/6000

The software support package is installed from diskettes on an AIX host system using the system management interface tool (**smit**).

Before beginning the installation, you must have:

- **EVB for RS/6000** installation diskettes
- RISC System/6000, running AIX Version 3.2.5 or higher
- Superuser privileges on the AIX system

The method used to perform Steps 7 through 20 of the installation procedure depends on your version of **smit**. To select options, use the appropriate method for your version.

- In the X Window version, position the cursor and make selections using the mouse.
- In the character-based version, position the cursor using arrow keys and make selections using function keys.

The following procedure installs the EVB software support package.

1. Log in as **root** or use the AIX **su** command to become the superuser.
2. Use a **cd** command to change to the directory where the install image file will be stored.

   Typically, the directory **/usr/sys/inst.images** holds install image files. However, any directory can be used.
3. Insert the EVB installation diskette labeled “1 of *n*” (*n* may vary) into the diskette drive.
4. Run the following **restore** command to read the file **EVB.instal.Z** from the diskette into the working directory.

   restore –f/dev/rfd0

5. Insert the rest of the EVB installation diskettes into the diskette drive when prompted.

6. After the diskettes are read, unpack the file.

   uncompress EVB.instal.Z

7. Run the following command to begin the installation via **smit**.

   smit install_latest

8. Type the fully qualified path name of the file **EVB.instal** into the **Input device/directory for software** field.

   The path includes the directory selected in Step 2, for example, **/usr/sys/inst.images/EVB.instal**.

9. Press **Enter**.

10. Position the cursor on the **Software to install** line.

11. Select the **list** button (X Window version) or the **F4=List** function key (character-based version) to display a list of available software.

12. From the list, select the item or items appropriate for your platform and application.

    - To install the IBM High C/C++ Compiler, select the highc base item (ELF file format version only).
    - To install the complete OS Open distribution, select both the OS Open base and the OS Open platform specific items.

13. Select **OK** to complete the selection process and return to the **Install Software Products at Latest Available Level** window.

14. Ensure that the response for **Automatically install PREREQUISITE software** is "**no**".

    For systems running AIX 4 or later, this field is called **AUTOMATICALLY install requisite software.**

15. Ensure that the response for **OVERWRITE existing version** is "**yes**".

    For systems running AIX 4 or later, this field is called **OVERWRITE same or newer versions.**

16. Ensure that the response for **COMMIT Software** is "**yes**".

    For systems running AIX 4 or later, this field is called **COMMIT software updates**.

17. Begin the installation by selecting **Do** or **OK**.

18. Select **OK** at the **ARE YOU SURE?** screen to continue the installation.

19. When the Command status is **OK**, file installation is complete.

20. Exit **smit**.

The IBM High C/C++ Compiler is installed in the **/usr/highcppc** directory tree and the EVB software support package in the **/usr/osopen** directory tree. It may be necessary to change ownership of these directories, their subdirectories and their contents if other users will require access to them. The **/usr/highcppc/bin** directory contains the files required for the IBM High C/C++ Compiler. Those files include:

- asppc - Assembler for assembler language programs
- ldppc - ELF linker/binder to build applications to be run on the EVB
- hcppc - High C/C++ compiler for C programs
- arppc - ELF library archiver

The **readme** file under the **/usr/highcppc** directory contains the latest information regarding the compiler and should be considered “must reading”.

If you installed the compiler into a directory other than **/usr/highcppc**, edit the **bin/hcppc.cnf** file, and locate the line near the top of the file that reads **HCDIR=/usr/highcppc**. Change this to reflect the directory that the compiler was installed into. Save your changes and exit the editor.

The **/usr/osopen** directory tree contains the files and tools that support OS Open application and ROM development. The **/usr/osopen** subdirectories and their contents are as follows.

- **/bin**

This directory contains several host based utilities used for application and ROM program development.

  - elf2rom - creates a ROM image from an ELF executable file
  - eimgbld - creates a ROM Monitor loadable image from an ELF executable file
  - hbranch - places an absolute branch in the last address of a ROM image
  - rambuild - creates an assembler source file that contains the files found in a specified directory
  - tracefmt - post-processes OS Open trace snapshots for AIX 3.2.X
  - trc41 - post-processes OS Open trace snapshots for AIX 4.1

- **/examples**

This directory contains many example OS Open programs.

- **/PLATFORM**

This directory contains the OS Open platform specific code for the platform included in your EVB kit. The directory is not literally named “PLATFORM”, but rather is named to identify the board and processor that was shipped with your kit. For example, if your platform was the 401GF evaluation board, this directory might be named m401_evb.

  - README.TXT - contains the latest information regarding this release
  - /include - contains OS Open include files

- /ld - contains dynamically loadable modules that can be run from OS Open's OpenShell
- /lib - contains OS Open libraries
- /m4 - contains assembler preprocessor include files
- /openbios - contains the source code for the ROM Monitor (detailed in a later chapter)
- /samples - contains samples programs that can be compiled and run

Considerable effort goes into providing a quality product with consistent documentation. To insure that our customers have the advantage of the latest software features and updated information, README.TXT may contain clarifications and/or additional information and should be considered "must reading".

- **/COMMENT.USR and COMMENT.DOC**

Please take the time to complete these user comment forms. Your feedback and suggestions will help us to improve our products and technical publications. Fax and email instructions are included in each of the files.

3.1.2 RISCWatch Debugger Installation - RS/6000

Please refer to the *RISCWatch Debugger User's Guide* for debugger installation instructions. Be sure to follow the instructions for RS/6000 installation.

3.2 PC Installation

3.2.1 EVB Software Support Package Installation - PC

Before beginning the installation, you must have:

- **EVB for PC** installation diskettes
- PC running Windows 3.1 or higher, Windows 95, or Windows NT 3.51

The following procedure installs the EVB software support package:

NOTE: For Windows NT users, we recommend that you logon as "root".

1. Insert the installation diskette labeled "EVB - PC" and "1 of *n*" (*n* may vary) into diskette drive A:
2. Start Microsoft Windows if it is not active
3. Select Run... from the File pull-down of Program Manager or from the Start menu for Win95/NT
4. Type 'A:INSTALL' to run the installation program
5. Follow the installation program instructions

Once completed, the IBM High C/C++ Compiler is installed in the **\highcppc** directory tree and the EVB software support package in the **\osopen** directory tree. The **\highcppc\bin** directory contains the files required for the IBM High C/C++ Compiler. Those files include:

- asppc.exe - Assembler for assembler language programs
- ldppc.exe - ELF linker/binder to build applications to be run on the EVB
- hcppc.exe - High C/C++ compiler for C programs
- arppc.exe - ELF library archiver

The **readme** file under the **\highcppc** directory contains the latest information regarding the compiler and should be considered “must reading”.

The **\osopen** directory tree contains the files and tools that support OS Open application and ROM development. The **\osopen** subdirectories and their contents are as follows.

- **\bin**

This directory contains several host based utilities used for application and ROM program development.

- elf2rom.exe - creates a ROM image from an ELF file
- eimgbld.exe - creates a ROM Monitor loadable image from an ELF executable file
- hbranch.exe - places an absolute branch in the last address of a ROM image
- rambuild.exe - creates an assembler source file that contains the files found in a specified directory
- make.exe - supports the use of makefiles when building application programs
- bootpd.exe - bootp server to support ROM Monitor downlaods
- tftpd.exe - tftp server to support host-to-EVB file transfers

- **\examples**

This directory contains many example OS Open programs.

- **\PLATFORM**

This directory contains the OS Open platform specific code for the platform included in your EVB kit. The directory is not literally named “PLATFORM”, but rather is named to identify the board and processor that was shipped with your kit. For example, if your platform was the 401GF evaluation board, this directory might be named m401_evb.

- README.TXT - contains the latest information regarding this release
- \include - contains OS Open include files
- \ld - contains dynamically loadable modules that can be run from OS Open’s OpenShell
- \lib - contains OS Open libraries
- \m4 - contains assembler preprocessor include files

- \openbios - contains the source code for the ROM Monitor (detailed in a later chapter)
- \samples - contains samples programs that can be compiled and run

Considerable effort goes into providing a quality product with consistent documentation. To insure that our customers have the advantage of the latest software features and updated information, README.TXT may contain clarifications and/or additional information and should be considered “must reading”.

- **\COMMENT.USER and \COMMENT.DOC**

Please take the time to complete these user comment forms. Your feedback and suggestions will help us to improve our products and technical publications. Fax and email instructions are included in each of the files.

3.2.2 RISCWatch Debugger Installation - PC

Please refer to the *RISCWatch Debugger User’s Guide* for debugger installation instructions. Be sure to follow the instructions for PC installation.

3.3 Sun Installation

3.3.1 EVB Software Support Package Installation - Sun

The software support package is installed from diskettes on a Sun host system using the **cpio** and **tar** commands.

Before beginning the installation, you must have:

- **EVB for Sun** installation diskettes
- a Sun SPARCstation 5. 10, or 20 workstation running SunOS 4.1.3 (or higher) or Solaris 2.3 (or higher)
- Superuser privileges on the Sun system

The procedures required for installing the EVB software support package vary depending on the operating system being used. Please follow the instructions corresponding to your operating system.

1. Log in as **root** or use the **su** command to become the superuser
2. Open at least two windows for this procedure
3. Use the **cd** command to change to the **/usr** directory
4. Insert the installation diskette labeled “EVB - Sun” and “1 of *n*” (*n* may vary) into the diskette drive.

Instructions for SunOS 4.1.3 (or higher) only:

5. From the second window run the command:

   cpio -ivB EVB_os4.tar.Z EVB.tar.Z EVB_hcppc.tar.Z < /dev/rfd0

   where **/dev/rfd0** is the name of your diskette device.

6. When the system prompts you for a new volume, move to the first window and type **eject** to eject the diskette. Insert the next diskette.

7. Move to the second window and type the name of the diskette drive (**/dev/rfd0)** to continue the process.

8. If prompted for more diskettes, repeat the previous two steps. When finished, type **eject** to remove the final diskette.

9. Return to the first window and verify that the following files are installed under the **/usr** directory:

   EVB.tar.Z

   EVB_os4.tar.Z

   EVB_hcppc.tar.Z

10. Run the following commands to unpack and install the files (**order is important**):

    zcat EVB.tar.Z | tar xvf -

    zcat EVB_os4.tar.Z | tar xvf -

    zcat EVB_hcppc.tar.Z | tar xvf -

Installation for SunOS is complete. The tar.Z files may be removed to recover space.

Instructions for Solaris 2.3 (or higher) only:

11. From the first window type **volcheck.** This creates a file called **/vol/dev/rdiskette0/unlabeled** (the diskette device name).

    If the system pops up a message box saying the diskette format is unrecognized, ignore the message and cancel the message box. The name of the file created may be different on your system. You can use the **eject -q** command to see the actual name. The file name returned is the name that should be used in the subsequent steps.

12. From the second window run the command:

    cpio -ivB EVB.tar.Z EVB_hcppc.tar.Z < /vol/dev/rdiskette0/unlabeled

    where **/voldev/rdiskette0/unlabeled** is the name of your diskette device.

13. When the system prompts you for a new volume, move to the first window. Type **eject** if the system did not automatically eject the diskette. Insert the next diskette and type **volcheck**.
14. Move to the second window and type the name of the diskette drive (**/vol/dev/rdiskette0/unlabeled) to continue the process**.
15. If prompted for more diskettes, repeat the previous two steps. When finished, type **eject** to remove the final diskette.
16. Return to the first window and verify that the following files are installed under the **/usr** directory.

    EVB.tar.Z

    EVB_hcppc.tar.Z

17. Run the following commands to unpack and install the files.

    zcat EVB.tar.Z | tar xvf -

    zcat EVB_hcppc.tar.Z | tar xvf -

Installation for Solaris is complete. The tar.Z files may be removed to recover space.

The IBM High C/C++ Compiler is installed in the **/usr/highcppc** directory tree and the EVB software support package in the **/usr/osopen** directory tree. It may be necessary to change ownership of these directories, their subdirectories and their contents if other users will require access to them. The **/usr/highcppc/bin** directory contains the files required for the IBM High C/C++ Compiler. Those files include:

- asppc - Assembler for assembler language programs
- ldppc - ELF linker/binder to build applications to be run on the EVB
- hcppc - High C/C++ compiler for C programs
- arppc - ELF library archiver

The **readme** file under the **/usr/highcppc** directory contains the latest information regarding the compiler and should be considered “must reading”.

If you installed the compiler into a directory other than **/usr/highcppc**, edit the **bin/hcppc.cnf** file, and locate the line near the top of the file that reads **HCDIR=/usr/highcppc**. Change this to reflect the directory that the compiler was installed into. Save your changes and exit the editor.

The **/usr/osopen** directory tree contains the files and tools that support OS Open application and ROM development. The **/usr/osopen** subdirectories and their contents are as follows.

- **/bin**

This directory contains several host based utilities used for application and ROM program

development.

- elf2rom - creates a ROM image from an ELF file
- eimgbld - creates a ROM Monitor loadable image from an ELF executable file
- hbranch - places an absolute branch in the last address of a ROM image
- rambuild - creates an assembler source file that contains the files found in a specified directory
- bootpd - bootp server to support ROM Monitor downlaods

- **/examples**

This directory contains many example OS Open programs.

- **/PLATFORM**

This directory contains the OS Open platform specific code for the platform included in your EVB kit. The directory is not literally named “PLATFORM”, but rather is named to identify the board and processor that was shipped with your kit. For example, if your platform was the 401GF evaluation board, this directory might be named m401_evb.

- README.TXT - contains the latest information regarding this release
- /include - contains OS Open include files
- /ld - contains dynamically loadable modules that can be run from OS Open’s OpenShell
- /lib - contains OS Open libraries
- /m4 - contains assembler preprocessor include files
- /openbios - contains the source code for the ROM Monitor (detailed in a later chapter)
- /samples - contains samples programs that can be compiled and run

Considerable effort goes into providing a quality product with consistent documentation. To insure that our customers have the advantage of the latest software features and updated information, README.TXT may contain clarifications and/or additional information and should be considered “must reading”.

- **/COMMENT.USER and /COMMENT.DOC**

Please take the time to complete these user comment forms. Your feedback and suggestions will help us to improve our products and technical publications. Fax and email instructions are included in each of the files.

3.3.2 RISCWatch Debugger Installation - Sun

Please refer to the *RISCWatch Debugger User’s Guide* for debugger installation instructions. Be sure to follow the instructions for Sun installation.

4

Host Configuration

Several host configuration steps are required to facilitate communications between the host computer and the evaluation board. These steps are outlined in this chapter. Please refer to the section corresponding to your host system.

4.1 RS/6000 Host Configuration

RS/6000 configuration requires that you be the superuser of the host workstation. This is accomplished by logging in as **root** or by using the AIX **su** command to become the superuser.

4.1.1 Serial Port Setup - RS/6000

The RS/6000 includes two serial ports to support communications via asynchronous data transfer. These ports are labeled S1 and S2 on the back of the RS/6000's system unit. When properly configured, one serial port can be used to connect a terminal emulator running on the host to the ROM Monitor running on the EVB, and the other to provide a **S**erial **L**ine **I**nternet **P**rotocol (or **SLIP)** network interface between the host and the EVB to download applications. This section addresses the proper configuration of the S1 and S2 serial ports to support these connections. Details on setting up the terminal emulator are discussed in a later chapter. In this section, S1 and S2 refer to the respective serial ports on the host RS/6000, and SP1 and SP2 (labeled J6 and J19 on the board) to the respective serial ports on the EVB.

The connection of the terminal emulator running on the host to the ROM Monitor running on the EVB, is made through the S1 serial port on the RS/6000 and the SP1 serial port on the EVB. A connection between the S2 serial port on the host and the SP2 serial port on the EVB, provides a SLIP network interface to download application programs from the host to the EVB. If the recommended Ethernet connection is going to be used, the S2-to-SP2 SLIP connection is optional and does not need to be established.

Proper setup involves the configuration of **tty** devices for both the S1 and S2 serial ports on the host. **tty0** is used for the terminal emulator-to-ROM Monitor connection and **tty1** for the host-to-EVB SLIP connection. It is also necessary to establish a SLIP network interface between S2 on the host and SP2 on the EVB. The following steps should be taken to insure proper S1, S2 configuration:

1. Log in as **root** or the superuser (**su**)

2. Determine if the **tty0, tty1** devices already exist

- enter **smit**
- select **Devices**
- select **TTY**
- select **List All Defined TTYs**

Perform step 3 for each tty not listed.

Perform step 4 for each tty listed to insure that it is properly configured.

3. To add a **tty** device

- return to the **TTY** screen
- select **Add a TTY**
- select **tty rs232 Asynchronous Terminal**
- select **sa0** - Serial Port 1 (for ROM Monitor connection) when adding **tty0**

  OR **sa1** - Serial Port 2 (for EVB SLIP connection) when adding **tty1**

- select **s1** for the port number when adding **tty0**
- OR **s2** for the port number when adding **tty1**
- insure that the BAUD rate is **9600** when adding **tty0**

  OR that the BAUD rate is **38400** when adding **tty1**

- insure that the PARITY is **none**
- insure that the BITS per character is **8**
- insure that the Number of STOP BITS is **1**
- insure that Enable LOGIN is **disabled**

  The default settings for all the other fields are satisfactory.

- select **Do** or hit **Enter**

  Upon successful completion, a properly configured **tty** device is created and thus, step 4 can be skipped for the particular **tty** (**tty0** or **tty1**) added. Remember to repeat this step, step 3, if both **tty0** and **tty1** needed to be added.

4. To properly configure a previously defined **tty** device

For systems running **AIX 3** :

- return to the **TTY** screen
- select **Change / Show Characteristics of a TTY**
- select **tty#** (where # = **0** or **1**)
- select **Change / Show TTY Program**
- insure that the following fields are set to the indicated values:

```
TTY                     tty#          (#=0 for tty0, 1 for tty1)
TTY type                tty
TTY interface           rs232
Description             Asynchronous Terminal
Status                  Available
Location                00-00-S*-00   (*=1 for tty0, 2 for tty1)
Parent Adapter          sa#          (#=0 for tty0, 1 for tty1)
Port Number             s*           (*=1 for tty0, 2 for tty1)
Terminal Type           dumb
Enable LOGIN            disable
```

The other fields can remain at their default values.

- select **Do** or hit **Enter**
- upon successful completion, select **Done** or hit **PF3** to return to the **TTY** screen
- select **Change / Show Characteristics of a TTY**
- select **tty#** (where # = **0** or **1**)
- select **Change/Show HARDWARE TTY Characteristics**
- insure that the BAUD rate is **9600** for **tty0**

  OR that the BAUD rate is **38400** for **tty1**

- insure that the PARITY is **none**
- insure that the BITS per character is **8**
- insure that the Number of STOP BITS is **1**
- select **Do** or hit **Enter**

Upon successful completion, the **tty** device is properly configured.

For systems running **AIX 4** or later :

- return to the **TTY** screen
- select **Change / Show Characteristics of a TTY**
- select **tty#** (where # = **0** or **1**)
- insure that the following fields are set to the indicated values:

```
TTY                     tty#          (#=0 for tty0, 1 for tty1)
TTY type                tty
TTY interface           rs232
Description             Asynchronous Terminal
Status                  Available
Location                00-00-S*-00   (*=1 for tty0, 2 for tty1)
Parent Adapter          sa#          (#=0 for tty0, 1 for tty1)
Port Number             s*           (*=1 for tty0, 2 for tty1)
Terminal Type           dumb
Enable LOGIN            disable
```

- insure that the BAUD rate is **9600** for **tty0**

  OR that the BAUD rate is **38400** for **tty1**

- insure that the PARITY is **none**
- insure that the BITS per character is **8**
- insure that the Number of STOP BITS is **1**

  The other fields can remain at their default values.

- select **Do** or hit **Enter**

  Upon successful completion, the **tty** device is properly configured.

5. This last step establishes the SLIP network over the **tty1 device** between the host and the EVB. It's optional for those using the recommended Ethernet connection for host-to-EVB communications. This step is **not** required for **tty0** since it is being used simply for terminal emulation. Unlike a LAN interface, a SLIP connection is point to point. We first need to specify an IP address for the host and then an IP address for the other end of the SLIP connection, which in this case, is the evaluation board. To do this:

   - enter **smit**
   - select **Communication Applications and Services**
   - select **TCP/IP**
   - select **Further Configuration**
   - select **Network Interfaces**
   - select **Network Interface Selection**
   - select **Add a Network Interface**
   - select **Add a Serial Line INTERNET Network Interfac**e
   - select **tty1**
   - set the INTERNET ADDRESS field to the host IP address. An acceptable value would be **8.1.1.4**
   - set the DESTINATION Address field to the evaluation board's IP address. An acceptable value would be **8.1.1.5**

     Make a note of the addresses selected for the host and the evaluation board. They will be needed later.

   - set the Network MASK to **255.255.240.0**
   - insure that ACTIVATE is **yes**
   - insure that the TTY PORT is **tty1**
   - leave the BAUD RATE field blank
   - leave the DIAL STRING field blank

- select **Do** or hit **Enter**

  Upon successful completion, the SLIP Network Interface is established over **tty1** and the serial port setup is complete.

  If this step fails, insure that a SLIP Network has not already been defined over **tty1**. To make this check, return to the **Network Interface Selection** screen in **smit** and select **List All Network Interfaces**. If **sl1** is listed then a network interface has already been defined for **tty1** and its characteristics may need to be changed. Return to the **Network Interface Selection** screen and select **Change/Show Characteristics of a Network Interface**. Select **sl1** and insure that the fields are set as stated previously in this step. (Note - there is no need to change the IP addresses in the INTERNET ADRRESS and DESTINATION Adress fields if they have already been defined, but use of the above mentioned IP addresses is strongly recommended to maintain consistency with the rest of the documentation.) Make a note of the IP addresses chosen since they will be needed later during board setup.

4.1.2 Ethernet Setup - RS/6000

In addition to (or in place of) the SLIP connection, an Ethernet connection can be used for host-to-EVB communications. The Ethernet connection is made through an Ethernet adapter on the host and the 10Base2 or 10BaseT connector on the EVB. Ethernet is much faster than SLIP and is recommended when downloading large applications on to the board or when using the RISCWatch debugger.

An Ethernet connection may require additional hardware. The 401 EVB supports connection via Standard Ethernet, thin coax (10Base2) or twisted pair (10BaseT).

A 10Base2 connection requires at least a thin coaxial cable and a BNC “T” connector when the EVB is added to an existing network. If the EVB is at one of the ends of the Ethernet network, a terminating resistor is also required. If the Ethernet network is exclusive between the host and the EVB, a thin coaxial cable, two BNC “T” connectors, and two BNC terminators are required. At a minimum, a 10BaseT connection requires a crossover Ethernet twisted pair cable (included in the kit) for point-to-point communications. If you want more than two nodes, you will need a hub and straight-through twisted pair cables.

Other hardware required will depend on the type of Ethernet adapter you have on your RS/6000 and whether the board is being connected to an existing Ethernet network. *AIX Communications Concepts and Procedures (GC23-2203, two volumes)* has additional information about the management and configuration of a TCP/IP network, including specifics as to how to configure an Ethernet network interface. Some of the basic steps are outlined below. You should consult your network administrator before attempting ethernet setup.

1. The host must be equipped to participate in a 10Base2 or 10BaseT Ethernet network. This may require the installation of an Ethernet adapter card for your specific RS/6000 model and, as discussed previously, additional connectivity hardware. Consult the documentation included with the hardware for installation instructions. Most RS/6000 models come with Ethernet adapters already installed. They are labeled ET in the back of the RS/6000 system unit.

2. Assuming the host system is equipped with the appropriate Ethernet adapter, the Ethernet interface must be configured properly. To do this:

    - log in as **root** or the superuser (**su**)
    - enter **smit**
    - select **Communication Applications and Services**
    - select **TCP/IP**
    - select **Further Configuration**
    - select **Network Interfaces**
    - select **Network Interface Selection**
    - select **Add a Network Interface**
    - select **Add a Standard Ethernet Network Interface**

      Note - choose "**Standard Ethernet**" as opposed to "IEEE 802.3 Ethernet". If you receive an error message stating that there is "No available adapter", go to step 3 and skip the remaining items in this step, step 2.

    - select **en0**
    - set the INTERNET ADDRESS field to the host IP address. This value must be different from that used for the SLIP interface. It can be set to any convenient value if the Ethernet network is private for 401 EVB development purposes. An acceptable value would be **7.1.1.4**

      Make a note of the IP address selected for the host system. It will be needed later. Note that an IP address for the evaluation board is not required as it was for the point-to-point SLIP network interface. An IP address for the EVB will, however, be required later on for the board setup.

    - set the Network MASK field to **255.255.240.0**
    - insure that ACTIVATE is **yes**
    - insure that the Use Address Resolution Protocol is **yes**
    - leave the BROADCAST ADDRESS blank
    - select **Do** or hit **Enter**

      Upon successful completion, a properly configured Ethernet interface has been added. The Ethernet setup is complete and step 3 need not be performed.

3. Perform this step only if you received the "No available adapter" error message when trying to **Add a Standard Ethernet Network Interface** in step 2. This message indicates that either the Ethernet adapter is missing (or possibly misplugged) or the Ethernet Network Interface already exists. To determine if the interface already exists:

    - return to the **Network Interface Selection** screen in **smit**

- select **Change/Show Characteristics of a Network Interface**

  If **en0** is **not** listed, insure that the RS/6000 host does have an Ethernet adapter and, if possible, that it is plugged correctly. If the adapter was misplugged, repeat step 2 to add the Ethernet Network Interface.

  if **en0** is listed, then the Ethernet Network Interface already exists. Select **en0** and note the IP address listed for the INTERNET ADDRESS field. This value is the host's Ethernet IP address and will be needed later. If no IP address is listed, choose one. The IP address **7.1.1.4** can be used to maintain consistency with the menus and examples in this document. The Ethernet setup is complete.

4.1.3 ROM Monitor-Debugger Communication Setup - RS/6000

Before the RISCWatch Debugger can be used, some additional steps need to be taken to establish ROM Monitor-Debugger communications. These steps involve an update of the TCP/IP **services** file and a refresh of the TCP/IP **inetd** daemon.

To modify the **/etc/services** file, you need to log in as **root** or the superuser (**su**). The following lines must be added to the file:

```
osopen-dbg    20044/tcp     #  for RISCWatch OS Open debug
osopen-dbg    20044/udp     #  for RISCWatch rom_mon debug
```

The AIX **refresh -s inetd** command must then be run to inform the **inetd** daemon of the changes made to the /etc/services file.

4.2 PC Host Configuration

As stated previously, PC users are required to have a TCP/IP package compliant with the Microsoft Windows Socket API definition. Unlike Windows 95 and Windows NT, Windows 3.1 does not include such a package. To determine if you will need to install a TCP/IP package on Windows 3.1, do the following:

- Select the **Main** icon from the Windows' Program Manager.
- Select the **File Manager** icon.
- Select **File** from the menu bar and choose **Search**.
- Perform a search for **winsock.dll** on your entire hard drive.

If the winsock.dll file exists, you probably have some compliant TCP/IP package already installed. **Workgroup for Windows** is a product that provides such a TCP/IP package. If the winsock.dll file does not exist, you need to install a TCP/IP package compliant with the Microsoft Windows Socket API definition. One such package, Trumpet Winsock, can be downloaded from the following Internet site: **www.trumpet.com**.

Note: Windows 95 users who want to establish a SLIP host-to-EVB connection over a second serial port, require Trumpet Winsock as well, since the TCP/IP package that comes with Windows 95 does not support SLIP communications. Trumpet is not recommended for Windows 95 users already connected to a network since installing Trumpet may cause problems with previously defined networks. If the recommended Ethernet host-to-EVB connection is going to be used (instead of the SLIP host-to-EVB connection), Windows 95 users do **not** need to install Trumpet since the TCP/IP package that comes with Windows 95 can be used to establish the Ethernet connection.

The following information is provided as a **guide** to installing the Trumpet Winsock code. It is not meant to be a replacement to the installation instructions contained at the Trumpet Internet site. It is provided to help clarify items which may be confusing.

1. Go to the Trumpet Software International's web site (http://www.trumpet.com) and find the installation information for Trumpet Winsock. You want to download the latest version which can be used for Windows 3.1 (must have 16 bit support). For example, version 3.0 (file twsk30c.exe) is a combined 16 bit/Windows 95 release. This version can be downloaded and used for an evaluation period of 30 days. Use beyond the evaluation period requires a purchase.
2. The downloaded version is usually a single file called a self extracting ZIP file (has an extension *.exe). This file should be installed in a new directory (c:\trumpet, for example) and then executed. Execution is accomplished by going to the newly created directory and entering the name of the file. This will result in the creation of many more files in the new directory.
3. Read any '**README**' files carefully. Ethernet users are interested in directions concerning Packet Drivers because you will not be using a modem and you have already determined that a TCP/IP package does not exist on your system.
4. If the readme file does not direct you to do otherwise, execute 'install.exe' to start the installation process. You will be prompted for any required information. Note that you may be informed that a search will be done to rename any '**winsock.dll**' files found. If you performed this check earlier, this file should not be found anywhere else on your hard drive.
5. If a 'setup' screen appears, you can defer entering any fields until a later time.
6. When installation is complete, reboot the system, and bring up Windows.

4.2.1 Serial Port Setup

Most PCs include two serial ports to support communications via asynchronous data transfer. These ports are sometimes referred to as communication or COM ports. These ports are usually accessed from the back of the system unit. This document refers to them as serial ports S1 and S2. You should consult your PC literature to determine how many serial ports are available on your unit and where they are located.

When properly configured, one serial port can be used to connect a terminal emulator running on the host to the ROM Monitor running on the EVB, and the other to provide a **S**erial **L**ine **I**nternet **P**rotocol (or **SLIP)** network interface between the host and the EVB to download applications. The SLIP host-to-EVB connection is optional if the recommended Ethernet connection is going to be used for host-to-EVB communications. This section addresses the proper configuration of the S1 and S2 serial ports to support these connections. Users should also refer to the Windows on-line help for “Changing Serial Port Settings”.

The connection of the terminal emulator running on the host to the ROM Monitor running on the EVB, is made through the S1 serial port on the PC and the SP1 serial port on the EVB. The S1 port must be configured for a baud rate of 9600, 8 data bits, 1 stop bit, and no parity. The proper setting of these parameters is discussed later in the section on terminal emulation.

A connection between the S2 serial port on the host and the SP2 serial port on the EVB, provides a SLIP network interface to download application programs from the host to the EVB. This connection can be used in place of or along with the recommended Ethernet connection.

To establish a SLIP network over the S2 serial port for host-to-EVB communications, define a SLIP interface via the TCP/IP package being used. Since TCP/IP packages for PCs vary, users should consult their TCP/IP literature or their system administrator on how to establish the SLIP interface between the host and the EVB. The following IP addresses are suggested for the SLIP interface:

- PC host (source) : **8.1.1.4**
- Board (destination) : **8.1.1.5**

Make a note of the IP addresses selected since they will be needed later.

Trumpet Winsock users can use the following steps as a guide to establishing the SLIP interface:

1. Open the Trumpet Winsock by double clicking on the Trumpet Winsock icon in the Trumpet Winsock Files program group.
2. If setup was bypassed during installation, your connection should fail. A Trumpet Winsock window comes up indicating your connection status. Select **Setup** from the File menu to open the Setup dialog.
3. Set the **IP address** field to the IP address of the PC host: **8.1.1.4** is suggested to maintain consistency with this document.
4. Select **SLIP** under Drivers and then go to **Dialler settings**.
5. Select the appropriate **COMM port** (COM2 for example) to be used for SLIP communications.
6. Set the **Baud rate** to **38400**.

7. Disable **Hardware handshaking** and make sure **No automatic login** is selected. Use the default settings for the remaining options and/or check the help for more details.
8. Select **OK** from **Dialler Settings** and then **OK** from **Setup**.
9. Edit the **hosts** file found in the installed Trumpet directory to include both the PC host IP address and the board IP address. For example:
   8.1.1.4 local_slip
   8.1.1.5 evb_slip

After entering all the information, you may need to restart Trumpet Winsock for the network setup to take effect.

Prior to exiting Windows, we recommend terminating Trumpet Winsock (close the application). If you do not follow this recommendation, subsequent Trumpet starts may fail. If this occurs, you will need to reboot your system.

4.2.2 Ethernet Setup - PC

In addition to (or in place of) the SLIP connection, an Ethernet connection can be used for host-to-EVB communications. The Ethernet connection is made through an Ethernet adapter on the host and the 10Base2 or 10BaseT connector on the EVB. Ethernet is much faster than SLIP and is recommended when downloading large applications on to the board or when using the RISCWatch debugger.

An Ethernet connection requires additional hardware. The 401 EVB supports connection via Standard Ethernet, thin coax (10Base2) or twisted pair (10BaseT). This connection requires that the host PC be equipped with an appropriate Ethernet adapter. The host adapter is not included in the EVB kit. Please consult your PC and adapter documentation for requirements and installation instructions.

A 10Base2 connection requires at least a thin coaxial cable and a BNC “T” connector when the EVB is added to an existing network. If the EVB is at one of the ends of the Ethernet network, a terminating resistor is also required. If the Ethernet network is exclusive between the host and the EVB, a thin coaxial cable, two BNC “T” connectors, and two BNC terminators are required. At a minimum, a 10BaseT connection requires a crossover Ethernet twisted pair cable (included in the kit) for point-to-point communications. If you want more than two nodes, you will need a hub and straight-through twisted pair cables.

Other hardware required will depend on the type of Ethernet adapter you have on your PC and whether the board is being connected to an existing Ethernet network. Please consult the documentation included with the adapter hardware for additional instructions.

Since TCP/IP packages for PCs vary, users should consult their TCP/IP documentation for information regarding the management and configuration of an Ethernet network interface. Establishment of an ethernet interface requires a host IP address. If the host PC is connected to an existing ethernet network, the host IP address should already be defined. Consult your network administrator on how to obtain the host’s ethernet IP address and how to add the EVB to the existing network.

To maintain consistency with this document, the following IP addresses are suggested for the Ethernet interface :

- PC host (source) : **7.1.1.4**
- Board (destination) : **7.1.1.5**

Make a note of the IP addresses selected since they will be needed later.

4.2.2.1 Windows 3.1

Trumpet Winsock users can use the following steps as a **guide** to establishing a local Ethernet interface:

1. Trumpet Software International provides software which works with 'packet drivers'. When you first install your ethernet card, a set of different device drivers are provided. In order to use Trumpet Winsock, you will need to select a 'Packet Driver'. The Kingston ethernet card, provided with some RISCWatch packages, contains a packet driver that can be selected. If you buy an ethernet card that does not contain a packet driver, you can use the help option on the Trumpet menu bar to find out how you may be able to obtain a packet driver from the Internet. We will assume you have already followed the instructions for installing your ethernet card, have installed Trumpet Winsock, and have chosen a packet driver for use with Trumpet.
2. Read any '**README**' files carefully. Pay particular attention to any directions concerning Packet Drivers.
3. Follow the instructions for **Using the Trumpet Winsock over a packet driver** from the main Trumpet Help window. Follow the instructions for **Installing a packet driver and WINPKT**. At the time of this publication, the WINPKT program needed to be extracted from 'ftp://ftp.trumpet.com/winsock/winpkt.com'. The ndis3pkt package, referred in the help as a replacement for winpkt, does not work unless you have WorkGroups for Windows, or some other windows package that runs NDIS.
4. Using the Trumpet help as a guide, your '**autoexec.bat**' file will need to have two lines added to get the ethernet communications working. The first line starts the packet driver you installed with your ethernet card. The proper name and syntax for this line should be identified in your ethernet card installation guide or in one of the files that came with the packet driver (i.e. the Kingston ethernet card has a '.doc' file that is part of the packet driver that describes how to invoke the driver).The second line to add is '**winpkt 0x60**' (vector 0x60 is usually the default vector to use).
5. After updating the 'autexec.bat' file, reboot the system to execute the changes.
6. From Windows, start Trumpet Winsock by double clicking on the Trumpet Winsock icon in the Trumpet Winsock Files program group.

7. If setup was bypassed during installation, your connection should fail. A Trumpet Winsock window comes up indicating your connection status. Select **Setup** from the File menu to open the Setup dialog.
8. Set the **IP address** field to the IP address of the PC host: **7.1.1.4** is suggested to maintain consistency with this document.
9. Select **Packet driver**, and set the **Vector** to **60**, **Netmask** to **255.255.240.0**, and **Gateway** to **0.0.0.0**.
10. Select **OK**.
11. Edit the **hosts** file found in the installed Trumpet directory to include both the PC host IP address and the board IP address. For example:
    7.1.1.4 local_enet
    7.1.1.5 evb_enet

After entering all the information, you may need to restart Trumpet Winsock for the network setup to take effect.

Prior to exiting Windows, we recommend terminating Trumpet Winsock (close the application). If you do not follow this recommendation, subsequent Trumpet starts may fail. If this occurs, you will need to reboot your system.

4.2.2.2 Ethernet Setup - Windows 95

A compliant TCP/IP package comes with Windows 95, so no TCP/IP package needs to be installed. If you haven't done so already, install the ethernet card on the host system according to the directions that came with the card.

To set the Host IP address for the ethernet connection:

- select the 'My Computer' icon from the desktop.
- select 'Control Panel'.
- select 'Network'.
- Add the appropriate "Adapter" network component for the ethernet adapter being used (if not already added).
- Add a "Protocol" network component of 'Microsoft - TCP/IP' (if not already added). Specify the IP address (**7.1.1.4** is recommended to maintain consistency with this document) and netmask (**255.255.240.0**) to be used.

Note: The "services" file that must be updated as part of the RISCWatch or evaluation kit installation is in directory "C:\WINDOWS".

The Evaluation Kit software was developed for Windows 3.1. Though it can be run successfully on Windows 95, certain restrictions apply. For example, file IDs need to be restricted to an eight character file name, and a three character file extension, or RISCWatch will not be able to locate source files.

4.2.2.3 Ethernet Setup - Windows NT 3.51

A compliant TCP/IP package comes with Windows NT, so no TCP/IP package needs to be installed. If you haven't done so already, install the ethernet card on the host system according to the directions that came with the card.

To configure TCP/IP for ethernet, double-click on the control panel icon followed by the network icon. Windows NT will prompt you through adding an ethernet adapter and TCP/IP. An IP address of **7.1.1.4** is recommended to maintain consistency with this document. A netmask of **255.255.240.0** should be used.

Note: The "services" file that must be updated as part of the RISCWatch or evaluation kit installation is in directory "C:\WINNT35\system32\drivers\etc".

The Evaluation Kit software was developed for Windows 3.1. Though it can be run successfully on Windows NT, certain restrictions apply. For example, file IDs need to be restricted to an eight character file name, and a three character file extension, or RISCWatch will not be able to locate source files.

4.2.3 ROM Monitor-Debugger Communication Setup - PC

Before the RISCWatch Debugger can be used, some additional steps need to be taken to establish ROM Monitor-Debugger communications. These steps involve an update of the TCP/IP **services** file and a restart of the TCP/IP package for the update to take effect.

Most PC TCP/IP packages place the **services** file under one of the TCP/IP package's subdirectories. Trumpet Winsock users should find the **services** file in the directory where the Trumpet files were installed. Windows 95 users should find the **services** file under "C:\WINDOWS\SERVICES". Windows NT users will find the **services** file under "C:WINNT35\system32\drivers\etc". Users should consult their TCP/IP documentation or system administrator if they can not locate the file. The following lines must be added to the file:

```
osopen-dbg    20044/tcp     #  for RISCWatch OS Open debug
osopen-dbg    20044/udp     #  for RISCWatch rom_mon debug
```

For the update to take effect, TCP/IP needs to be re-started. This may require a re-boot of the system and/or a restart of the TCP/IP package.

4.3 Sun Host Configuration

Sun configuration requires that you be the superuser of the host workstation. This is accomplished by logging in as **root** or by using the **su** command to become the superuser.

4.3.1 Serial Port Setup - SUN

The Sun workstation includes two serial ports to support communications via asynchronous data transfer. These ports are labeled Serial A and Serial B on the back of the Sun's system unit. Some SPARCstation models multiplex these two ports into one physical port labeled A/B (Use A if it's available since use of the B port requires a special de-multiplexing cable from Sun). This section refers to these ports as S1 and S2, respectively. When properly configured, one of the serial ports can be used to connect a terminal emulator running on the host to the ROM Monitor running on the EVB. This connection is made through the S1 serial port on the Sun and the SP1 serial port on the EVB.

The S1 port on the host must be configured for a baud rate of 9600, 8 data bits, 1 stop bit, and no parity. The proper setting of these parameters is discussed later in the section on terminal emulation.

4.3.2 Ethernet Setup - SUN

Since all Sun SPARCstations come equipped with an ethernet (or AUI) port, an ethernet connection is used for host-to-EVB communications. The ethernet connection is made through the ethernet port on the host and the 10Base2 or 10BaseT connector on the EVB.

An Ethernet connection requires additional hardware. The 401 EVB supports connection via Standard Ethernet, thin coax (10Base2) or twisted pair (10BaseT).

A 10Base2 connection requires at least a thin coaxial cable and a BNC "T" connector when the EVB is added to an existing network. If the EVB is at one of the ends of the Ethernet network, a terminating resistor is also required. An exclusive Ethernet network between the host and the EVB, requires a thin coaxial cable, two BNC "T" connectors, and two BNC terminators. Depending on your SPARCstation model and options, an AUI (or thick ethernet) adapter cable or an AUI/Audio Adapter may also be necessary. Both of these cables are available from Sun. At a minimum, a 10BaseT connection requires a crossover Ethernet twisted pair cable (included in the kit) for point-to-point communications. If you want more than two nodes, you will need a hub and straight-through twisted pair cables. Consult the documentation included with the hardware for additional information.

Establishment of an ethernet interface requires a host IP address. If the host SPARCstation is connected to an existing ethernet network, the host IP address should already be defined. Consult your network administrator on how to obtain the host's ethernet IP address and how to add the EVB to the existing network. Make a note of the host's IP address since it will be needed later.

If the host SPARCstation is not connected to an existing ethernet network, then a network between the EVB and the host must be established. The **ifconfig** command can be used to establish such a network. Users should consult their network administrator and Sun documentation for additional information. A host IP address of 7.1.1.4 is suggested to maintain consistency with this document. Make a note of the IP address selected since it will be needed later during board setup.

4.3.3 ROM Monitor-Debugger Communication Setup - SUN

Before the RISCWatch Debugger can be used, the TCP/IP **services** file must be updated to allow ROM Monitor-Debugger communications.

To modify the **/etc/services** file, you need to log in as **root** or the superuser (**su**). The following lines must be added to the file:

```
osopen-dbg    20044/tcp     #  for RISCWatch OS Open debug
osopen-dbg    20044/udp     #  for RISCWatch rom_mon debug
```

5

401 EVB Connectors

This chapter describes the 401 EVB connectors. The 401 EVB can be accessed through two serial ports, an Ethernet 10Base2 or 10BaseT connector, a parallel port, and the RISCWatch JTAG connection. A Squall expansion interface is provided for connection to an external customer-supplied prototyping area, and two standard PC AT-type connectors are provided for power supply connection. A processor expansion interface for a logic analyzer and a PCI slot for PCI adapters are also provided.

Positions of the connectors and jumpers on the EVB are indicated on Figure 5-1.

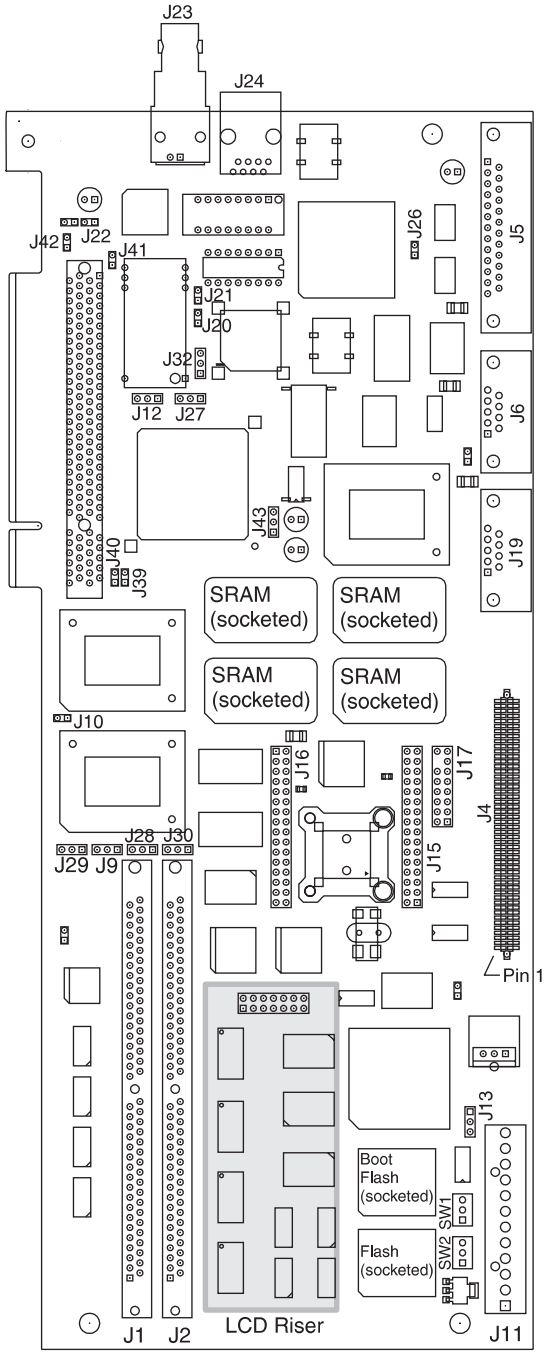


Figure 5-1. 401EVB Connectors and Jumpers

5.1 Serial Port Connectors

Serial ports 1 (J6) and 2 (J19) are provided with standard nine-pin male right-angle connectors, as shown in Figure 5-2 below:

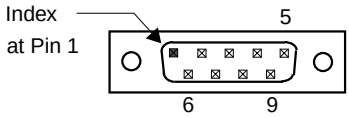


Figure 5-2. Nine-Pin Serial Port Connector

Table 5-1 describes the signal-to-pin assignments for serial ports 1 and 2:

Table 5-1. Serial Port Signal Assignments

Serial Port 1 (J6)		Serial Port 2 (J19)	
Pin Number	**Signal Name**	**Pin Number**	**Signal Name**
1	DCD	1	DCD
2	RX	2	RX
3	TX	3	TX
4	DTR	4	DTR
5	GND	5	GND
6	DSR	6	DSR
7	RTS	7	RTS
8	CTS	8	CTS
9	RI	9	RI

5.2 Ethernet Connectors

The 401 EVB is provided with a standard 10Base2 thin coax Ethernet connector (J23), and a standard 8-pin RJ-45 connector (J24) for 10BaseT. The RJ-45 connector is shown in Figure 5-2 below.

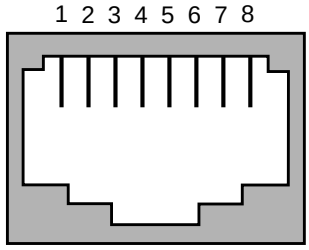


Figure 5-3. RJ-45 Ethernet Connector (Front View)

Table 5-3 describes the 10BaseT Ethernet connector signals on the 401 EVB and the recommended mating connectors:

Table 5-2. Ethernet Connector Description

Pin Number	Signal Name	Description
1	TX+	Transmit Data +
2	TX–	Transmit Data –
3	RX+	Receive Data +
4,5	NC	No Connection
6	RX–	Receive Data –
7,8	NC	No Connection

Table 5-3. Ethernet Connector Descriptions

Connector Type	Receptacle Specifications	Mating Connector Specifications
10Base-T	RJ45 Right-Angle modular jack, AMP 555164-1	AMP 554739-1, 554169-1, or 5541170-1, Molex 90075-0130 or 90075-0132
10Base2	Right-Angle BNC receptacle, AMP 227161-9	AMP 221128-1, 227079-5, 2-329082-1 or Molex 73100-5001, 73103-5001, or 73106-5001

5.3 Parallel Port Connector

The parallel port is provided with a standard D-type receptacle, shown in Figure 5-4 below.

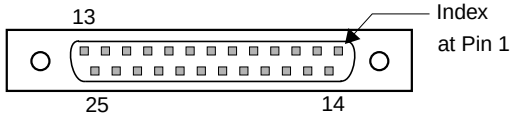


Figure 5-4. Parallel Port Connector (J5)

The pinout of the J5 parallel port connector is as follows:

Table 5-4. Connector J5 — Parallel Port

Pin	Name	Comment
1	$\overline{\text{STROBE}}$	
2 3 4 5 6 7 8 9	Data0 Data1 Data2 Data3 Data4 Data5 Data6 Data7	
10	$\overline{\text{ACK}}$	
11	BUSY	

Table 5-4. Connector J5 — Parallel Port (Continued)

Pin	Name	Comment
12	OUT	
13	SEL	
14	$\overline{\text{FEED}}$	
15	$\overline{\text{ERR}}$	
16	$\overline{\text{INIT}}$	
17	$\overline{\text{SLIN}}$	
18 19 20 21 22 23 24 25	GND GND GND GND GND GND GND GND	

5.4 RISCWatch JTAG Debugger Connectors

The RISCWatch JTAG debugger connects to the 401 EVB JTAG port (J17) through a 2 × 8-pin header. This header is shown in Figure 5-5:

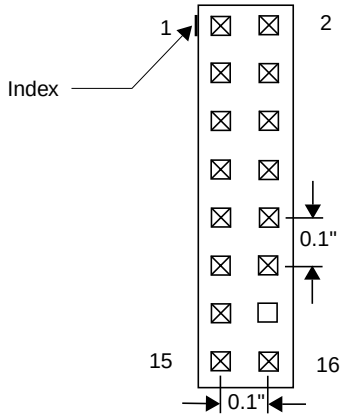


Figure 5-5. RISCWatch JTAG Header (Top View)

Placement of the RISCWatch JTAG header on the 401 EVB is indicated on the layout drawing in Figure 5-1 above. Signal names and positions on the headers are indicated in the following tables:

Table 5-5. RISCWatch JTAG Interface Connections and Resistors

Header Pin #	I/O	Signal Name	Description	401 Pin[1]	Board Resistor[2]
1	Out	TDO	JTAG test data out	1	
2			To be left unconnected		
3	In	TDI	JTAG test data in	8	10KΩ PU
4			To be left unconnected		
5			To be left unconnected		
6		+POWER[3]	Power (status signal, not processor V_{DD})		1KΩ PU[4]
7	In	TCK	JTAG test clock	18	10KΩ PU
8			To be left unconnected		
9	In	TMS	JTAG test mode select	17	10KΩ PU
10			To be left unconnected		
11	In	$\overline{\text{HALT}}$	Processor halt	16	10KΩ PU
12			To be left unconnected		
13			To be left unconnected		

Table 5-5. RISCWatch JTAG Interface Connections and Resistors

Header Pin #	I/O	Signal Name	Description	401 Pin[1]	Board Resistor[2]
14		KEY	Pin in this position should be removed.		
15			To be left unconnected		
16		GND	Ground		

[1]Pin numbers for PQFP packages

[2]PU = pullup resistor

5.5 Expansion/Test Interface Connector

Connectors J15 (processor expansion 1) and J16 (processor expansion 2) provide user access to all of the CPU control, address, and data signals.

Table 5-6 describes the signal assignments for the J15 processor expansion receptacle on the 401 EVB.

Table 5-6. Connector J15 — Processor Expansion 1

Pin	Name	Comment
1	+5V	
2	MemClk	
3	GND	
4	$\overline{\text{Crit_Int}}$	
5	Int	
6	GND	
7	HoldAck	
8	HoldReq	
9	+5V	
10	GND	
11	BusReq_401	
12	GND	

Table 5-6. Connector J15 — Processor Expansion 1 (Continued)

Pin	Name	Comment
13	$\overline{\text{Ready}}$	
14	$\overline{\text{BusErr}}$	
15	$\overline{\text{Reset}}$	
16	GND	
17 18	LA2 LA3	
19	$\overline{\text{ALE}}$	
20	GND	
21 22 23 24	BE3 BE2 BE1 BE0	
25	GND	
26	$\overline{\text{BLast}}$	
27	GND	
28	W/$\overline{\text{R}}$	
29	GND	
30 31	BusWidth0 BusWidth1	
32	$\overline{\text{Halt}}$	

Table 5-7 describes the signal assignments for the J16 processor expansion receptacle on the 401 EVB.

Table 5-7. Connector J16 — Processor Expansion 2

Pin	Name	Comment
1 2 3 4 5 6 7 8	B0 B1 B2 B3 B4 B5 B6 B7	401GF Address and Data. Address LSB. Data LSB in Little Endian; Data MSB in Big Endian. B0 is MSb of this byte.
9 10 11 12 13 14 15 16	B8 B9 B10 B11 B12 B13 B14 B15	401GF Address and Data. B8 is MSb of this byte.
17 18 19 20 21 22 23 24	B16 B17 B18 B19 B20 B21 B22 B23	401GF Address and Data. B16 is MSb of this byte.
25 26 27 28 29 30 31 32	B24 B25 B26 B27 B28 B29 B30 B31	401GF Address and Data. Address MSB. Data MSB in Little Endian; Data LSB in Big Endian. B24 is MSb of this byte.

5.6 Squall Expansion Interface

The EVB uses a 100-pin receptacle as the Squall interface connector, shown in Figure 5-6 below:

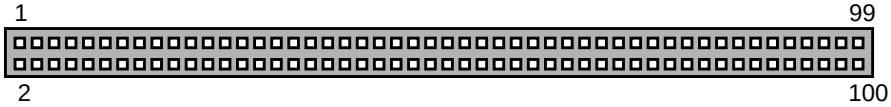


Figure 5-6. 100-Pin Squall Receptacle

Table 5-8 describes the expansion interface connector on the 401 EVB

Table 5-8. Connector J4 — Squall Interface

Pin	Name	Comment
1	S_ADS	Address Strobe
2	GND	
3	PMCLK	CPU Output Clock
4	GND	
5	$\overline{\text{S_BLAST}}$	Burst Last
6	$\overline{\text{S_LOCK}}$	Bus Lock
7	S_W/$\overline{\text{R}}$	Write/Read
8	GND	
9	$\overline{\text{S_READY}}$	
10	$\overline{\text{RESET}}$	
11 12 13 14	$\overline{\text{S_BE0}}$ $\overline{\text{S_BE1}}$ $\overline{\text{S_BE2}}$ $\overline{\text{S_BE3}}$	Byte Enables 32 bit: 3--24:31 ... 0--0:7 16 bit: 3--8:15, 2--nc, 1--A1, 0--0:7 8 bit: 3--nc, 2--nc, 1--A1, 0--A0
15	$\overline{\text{SQSEL}}$	Select Squall
16	GND	

Table 5-8. Connector J4 — Squall Interface (Continued)

Pin	Name	Comment
17 18	$\overline{\text{SQIRQ1}}$ $\overline{\text{SQIRQ0}}$	Interrupt Request
19	$\overline{\text{SQBR}}$	Shared Bus Request
20	$\overline{\text{SQBG}}$	Shared Bus Grant
21	$\overline{\text{S_EXTEND}}$	
22	GND	
23	+5V	
24 25 26 27 28 29 30 31	S_D31 S_D30 S_D29 S_D28 S_D27 S_D26 S_D25 S_D24	S_D31 is data MSb
32	GND	
33 34 35 36 37 38 39 40	S_D23 S_D22 S_D21 S_D20 S_D19 S_D18 S_D17 S_D16	
41	GND	

Table 5-8. Connector J4 — Squall Interface (Continued)

Pin	Name	Comment
42 43 44 45 46 47 48 49	S_D15 S_D14 S_D13 S_D12 S_D11 S_D10 S_D9 S_D8	
50	GND	
51 52 53 54 55 56 57 58	S_A31 S_A30 S_A29 S_A28 S_A27 S_A26 S_A25 S_A24	S_A31 is address MSb
59	+5V	
60 61 62 63 64 65 66 67	S_A23 S_A22 S_A21 S_A20 S_A19 S_A18 S_A17 S_A16	
68	+5V	
69 70 71 72 73 74 75 76	S_A15 S_A14 S_A13 S_A12 S_A11 S_A10 S_A9 S_A8	

Table 5-8. Connector J4 — Squall Interface (Continued)

Pin	Name	Comment
77 78 79 80 81 82	S_A7 S_A6 S_A5 S_A4 S_A3 S_A2	
83	+5V	
84 85 86 87 88 89 90 91	S_D7 S_D6 S_D5 S_D4 S_D3 S_D2 S_D1 S_D0	S_D0 is data LSb
92	GND	
93	+5V	
94 95		N/C N/C
96	GND	
97	SQSDA	
98	SQSCL	
99	+12V	
100	-12V	

5.7 Power Connector

The 401 EVB comes with two standard PC AT-type connectors for quick connect/disconnect to a power supply. A power supply and a line cord are provided with the 401 EVB. Power supply tolerances are ± 5% for the 5V supply. Table 5-9 defines the connections for the J11 power supply connector:

Table 5-9. Connector J11 — Power

Pin	Name	Comment
1	Power_Good	standard PC power supply: planar connector P1 to 1-6
2	+5V	
3	N/C	
4	N/C	
5	GND	
6	GND	
7	GND	standard PC power supply: planar connector P2 to 7-12
8	GND	
9	N/C	
10	+5V	
11	+5V	
12	+5V	

Warning: Use only the power supply provided in your EVB kit.

5.8 PCI Related Connectors

Table 5-10. Connector J39 — Accessory +3.3V

Pin	Name	Comment
1	+3.3V	standalone board: connect J39-1 to J40-1 to provide PCI socket +3.3V from board board operating as PCI adapter card: do not connect J39 to J40
2	GND	

Table 5-11. Connector J40 — PCI Socket +3.3V

Pin	Name	Comment
1	+3.3V	standalone board: connect J39-1 to J40-1 to provide PCI socket +3.3V from board; or connect external power supply +3.3V and GND to J39 board operating as PCI adapter card: do not connect to J40 (+3.3V to PCI socket is provided from PCI edge connector)
2	GND	

Table 5-12. Connector J41 — PCI Socket and Squall +12V

Pin	Name	Comment
1	+12V	standalone board: connect external power supply +12V and GND to J41 board operating as PCI adapter card: do not connect to J41 (+12V to PCI socket and Squall is provided from PCI edge connector)
2	GND	

Table 5-13. Connector J42 — PCI Socket and Squall -12V

Pin	Name	Comment
1	-12V	standalone board: connect external power supply -12V and GND to J42 board operating as PCI adapter card: do not connect to J42 (-12V to PCI socket and Squall is provided from PCI edge connector)
2	GND	

5.9 Setting the EVB Jumpers

The jumpers provided on the 401 EVB are detailed in the following tables:

Table 5-14. Jumper Programming of SIMM Size

PD2 (J30)	PD1 (J28)	PD0 (J9)	SIMM Size	Factory Setting
2-3 = lo	2-3 = lo	2-3 = lo	256Kx32 single-sided (1MB)	
2-3 = lo	2-3 = lo	1-2 = hi	512Kx32 double-sided (2MB)	
2-3 = lo	1-2 = hi	2-3 = lo	1Mx32 single-sided (4MB)	←
2-3 = lo	1-2 = hi	1-2 = hi	2Mx32 double-sided (8MB)	
1-2 = hi	2-3 = lo	2-3 = lo	4Mx32 single-sided (16MB)	
1-2 = hi	2-3 = lo	1-2 = hi	8Mx32 double-sided (32MB)	

Table 5-15. Jumper J9 — DRAM Programming PD0

J9	Description	Factory Setting
1-2	see Table 5-14 (Jumper Programming of SIMM Size) on page 17	
2-3		←

Table 5-16. Jumper J10 — Arbiter Select

J10	Description	Factory Setting
open	standalone board operation	←
1-2	board operating as PCI adapter card	

Table 5-17. Jumper J12 — PCI SERR Control

J12	Description	Factory Setting
open	standalone board; PCI socket SERR will interrupt the 401 as Critical Interrupt; locally generated PCI SERR will not generate an interrupt	
1-2	standalone board; PCI socket SERR will interrupt the 401 as Critical Interrupt; locally generated PCI SERR will interrupt the 401 as Critical Interrupt	←
2-3	board operating as PCI adapter card; local PCI SERR sent to PCI bus	

Table 5-18. Jumper J20 — Coax / Twisted-pair Selection

J20	Description	Factory Setting
open	coax selected	
1-2	twisted-pair selected	←

Table 5-19. Jumper J21 — Twisted-pair Link Integrity Checking

J21	Description	Factory Setting
open	enable twisted-pair link integrity checking	←
1-2	disable twisted-pair link integrity checking (jumper J22 should be open)	

Table 5-20. Jumper J22 — Twisted-pair Good-link LED

J22	Description	Factory Setting
open	LED disabled (should be open if J21 is closed)	
1-2	LED operational	←

Table 5-21. Jumper J26 — Parallel Port PP Mode

J26	Description	Factory Setting
open	bi-directional parallel port	←
1-2	uni-directional parallel port	

Table 5-22. Jumper J27 — PCI IDSEL Control

J27	Description	Factory Setting
1-2	standalone board; PCI bus AD16 used for local IDSEL	←
2-3	board operating as PCI adapter card; PCI bus IDSEL line used for local IDSEL	

Table 5-23. Jumper J28 — DRAM Programming PD1

J28	Description	Factory Setting
1-2	see Table 5-14 (Jumper Programming of SIMM Size) on page 17	←

Table 5-23. Jumper J28 — DRAM Programming PD1 (Continued)

J28	Description	Factory Setting
2-3		

Table 5-24. Jumper J29 — 25/33 MHz Bus Frequency Selection

J29	Description	Factory Setting
1-2	33 MHz	
2-3	25 MHz	←

Table 5-25. Jumper J30 — DRAM Programming PD2

J30	Description	Factory Setting
1-2	see Table 5-14 (Jumper Programming of SIMM Size) on page 17	
2-3		←

Table 5-26. Jumper J32 — PCI INTA Control

J32	Description	Factory Setting
open	standalone board; PCI socket INTA, INTB, INTC, INTD will interrupt the 401 as PCI INT; locally generated PCI INTA will not generate an interrupt	
1-2	standalone board; PCI socket INTA, INTB, INTC, INTD will interrupt the 401 as PCI INT; locally generated PCI INTA will interrupt the 401 as PCI INT	←

Table 5-26. Jumper J32 — PCI INTA Control (Continued)

J32	Description	Factory Setting
2-3	board operating as PCI adapter card; local PCI INTA sent to PCI bus	

Table 5-27. Jumper J43 — PCI Clock Source

J43	Description	Factory Setting
1-2	standalone board; board sources clock to PCI bus	←
2-3	board operating as PCI adapter card; board receives PCI clock from PCI bus	

5.10 Connecting the 401 EVB Hardware

5.10.1 Serial Port Connection

To establish a working environment, the EVB must be connected to a host system. ROM Monitor access requires a connection between the J6 serial port on the board and the S1 (COM1) serial port on the host. Users must also establish a connection for debug and

downloading applications from the host to the board. This connection is made over the SLIP or Ethernet network established during host configuration.

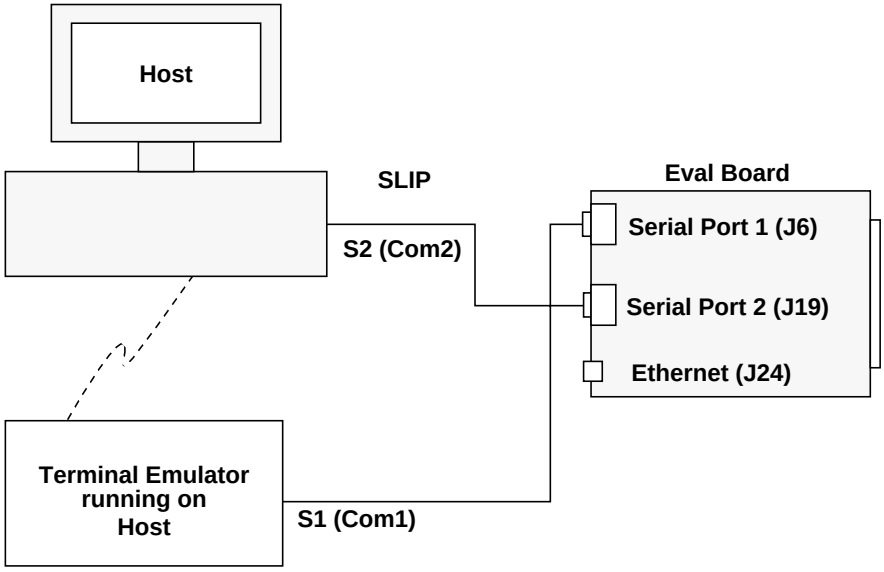


Figure 5-7. Serial Port Connection

Included in the 401 EVB kit is an interface cable supporting either 9-pin or 25-pin serial port connections. Use either the 9-pin or 25-pin, depending on the type of connector on the host. The cable is for connecting the J6 serial port 1 on the board to a terminal (or to a host running a terminal emulator). The board supports a second serial connection for communication over SLIP. This requires another interface cable (not provided) to attach to serial port 2 (J19).

Assuming a terminal emulator running on the host is going to be used for ROM Monitor access, connect the 9-pin serial port connector on one end of a cable to the J6 serial port on the EVB, and the other end of the same cable to the S1 (COM1) serial port on the host. The host end may require the 25-pin connector or a serial port adapter (not supplied) for connectivity. Sun SPARCstation users may require the 25 pin male-to-male adapter (included in the Sun 401 EVB kit) at the host end. If a SLIP connection is going to be used for host-to-EVB communications, connect a second cable (not provided) in a similar manner using J19 on the EVB and the S2 (COM2) serial port on the host.

5.10.2 Ethernet Connection

The Ethernet connection is made between the 10Base2 (J23) or 10BaseT (J24) connector on the 401 EVB and the Ethernet adapter on the host system. The board comes factory-set for 10BaseT communications and a 10BaseT crossover cable is included in the kit.

If the 10BaseT connection is to be used exclusively between the host and the EVB, the provided crossover cable can be used to directly connect the two nodes. Otherwise, a 10BaseT hub (not provided) must be used to connect the nodes together.

Note: The Ethernet 10BaseT crossover cable supplied will not work if plugged into a 10BaseT hub.

Figure 5-8 shows the connections and signal assignments required in a crossover cable:

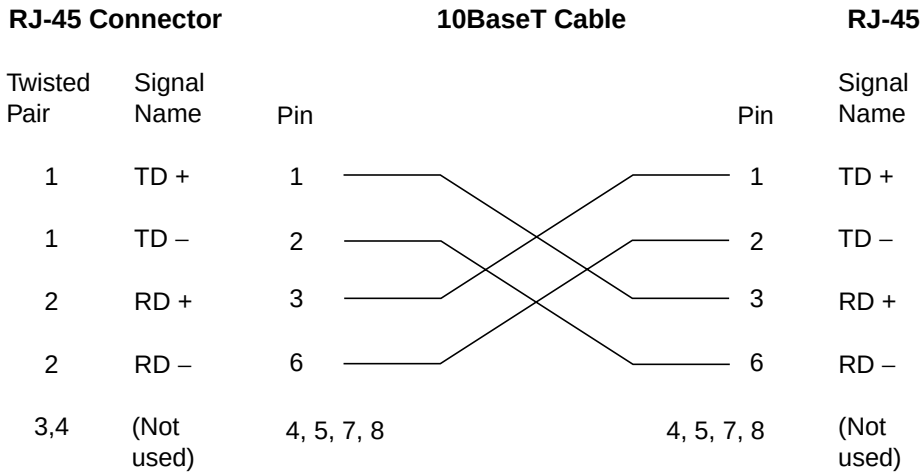


Figure 5-8. Wiring in a Crossover Cable

Figure 5-9 shows a point-to-point Ethernet connection using the provided crossover cable:

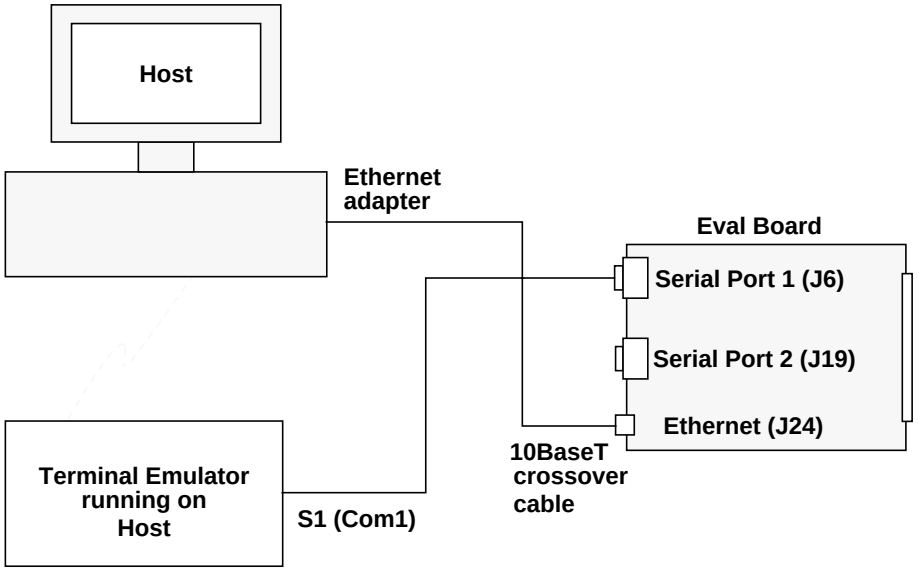


Figure 5-9. Point-to-Point 10BaseT Ethernet Connection

Figure 5-9 shows an Ethernet connection using a hub:

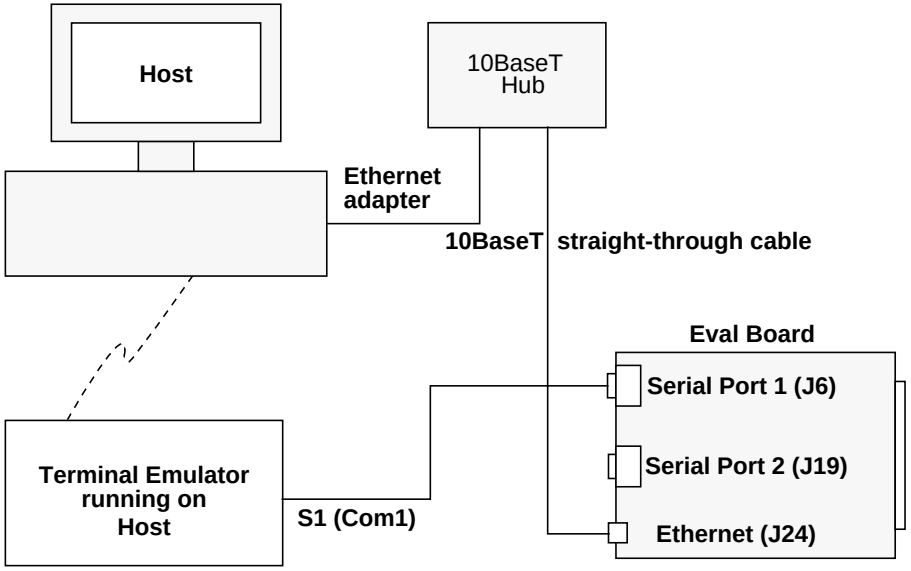


Figure 5-10. 10BaseT Ethernet Connection with Hub

If you wish to use **10Base2**, the board **jumper J20 must be left open** and additional connectivity hardware (not included in the kit) is required. The board should be connected to the host system via a 50 ohm thin coax cable. If the connection is to be used exclusively for communications between the host and the EVB, each end **must** have a BNC “T” type connector terminated at one end with a 50 ohm resistor. If the connection is going to be made to an existing Ethernet network, users should consult their Network Adminstrator to insure proper connectivity.

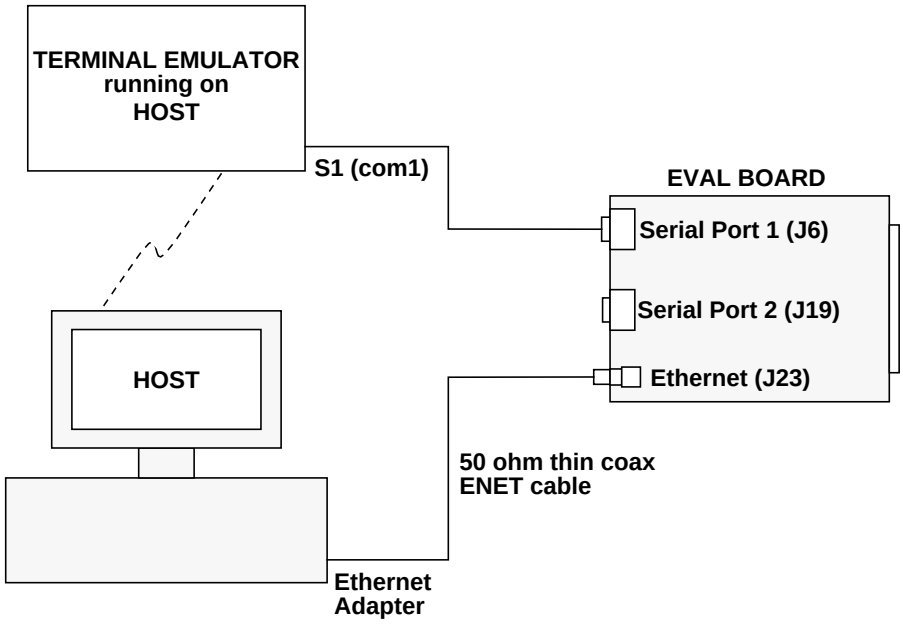


Figure 5-11. Point-to-Point 10Base2 Ethernet Connection

Note: Both a SLIP and Ethernet connection can be used as long as both networks have been configured properly and the proper connections have been made.

5.10.3 Power Supply Connection

Also included in the 401 EVB kit is a power supply and its power cord. Connect the female end of the power cord to the male connector on the power supply. Also connect the two keyed PC-AT type connectors (P1 and P2) from the power supply to the J11 power connector on the board.

5.11 Using a Terminal Emulator

The ROM Monitor transmits/receives data through serial port 1 (J6) on the evaluation board. Access to the ROM Monitor can be achieved by connecting a VT100 (or compatible) teminal

directly to J6 on the EVB or by using a terminal emulator running on the host. When using a terminal emulator, access is obtained via an S1(or COM1)-to-J6 connection.

5.11.1 RS/6000 Terminal Emulation

The AIX Terminal Interface Program (TIP) can be used as a terminal emulator to support communications with the ROM Monitor. When properly configured, TIP connects the host RISC/6000 to a remote system, which in our case is the EVB. To set up TIP, do the following:

- log in as **root** or the superuser (**su**)
- go to the **/etc** directory (**cd /etc)**
- see if the file, **remote**, exists (**ls remote).** If the file does **not** exist, create it.
- using an editor, add the following line to the **remote** file (cut and pasters can find this line in the README.TXT file) :

  tty0:dv=/dev/tty0:br#9600:el=^U^C^S^Q^D:ie=%$:oe=^D:pa=none:

- exit from **root**

TIP configuration is complete. Once all the host-to-EVB connections have been properly made and power has been supplied to the board, TIP can be activated by typing **tip tty0** at the AIX command prompt. After resetting the board, the ROM Monitor main menu should appear in the window where tip was activated. It may be necessary to hit the enter key once or twice to get the menu to appear for the first time. Additional information on TIP can be found in *AIX Communications and Procedures (GC23-2203, two volumes)*.

Some useful escape sequences to know when using TIP include (Note - it may be necessary to hit the **Enter** key before entering these escape sequences.):

- **~?** - help for TIP
- **~CTRL-D** - instructs the TIP command to terminate the connection and exit
- **~#** - sends a break to the remote system
- **~s script** - starts recording of transmissions made by the remote system

  Recordings are made in the default **tip.record** file in the user's current directory

- **~s !script** - stops recording of transmissions made by the remote system

Note - If a terminal emulator other than TIP is used, it must be configured for 9600 baud, eight bits per character, one stop bit, and no parity.

5.11.2 PC Terminal Emulation

Once all the host-to-EVB connections have been properly made and power has been supplied to the board, the Windows Terminal program can be used as a terminal emulator to support communications with the ROM Monitor. To do this:

- from Windows Program Manager, select Accessories
- select Terminal
- select Settings

- select Communications
- select COM1 (or the appropriate COM port used for S1 serial port set-up)
- select Baud Rate **9600**, Data Bits **8**, Stop Bits **1**, Parity **None**
- select Flow Control **Xon/Xoff**
- select OK

After resetting the board, the ROM Monitor menu should appear in the Terminal window. If it does not, check for proper connectivity between the host and the board. If the ROM Monitor menu still does not appear, insure that the COM port has been properly enabled. This can be done by using the configuration utility on the host PC (see your PC documentation for more details). Upon exiting the terminal program, you can save your setting in a ***.trm** file (for example, evb.trm) for future use.

5.11.3 SUN Terminal Emulation

The Terminal Interface Program (TIP) can be used as a terminal emulator to support communications with the ROM Monitor. When properly configured, TIP connects the host Sun SPARCstation to a remote system, which in our case is the EVB. To set up TIP, do the following:

- log in as **root** or the superuser (**su**)
- go to the **/etc** directory (**cd /etc)**
- see if the file, **remote**, exists (**ls remote).** If the file does **not** exist, create it.
- using an editor, add the following line to the **remote** file (cut and pasters can find this line in the README.TXT file) :

  tty0:dv=/dev/ttya:br#9600:el=^U^C^S^Q^D:ie=%$:oe=^D:pa=none:

- **exit** from root

TIP configuration is complete. Once all the host-to-EVB connections have been properly made and power has been supplied to the board, TIP can be activated by typing **tip tty0** at the command prompt. After resetting the board, the ROM Monitor main menu should appear in the window where tip was activated. It may be necessary to hit the enter key once or twice to get the menu to appear for the first time. If the ROM Monitor menu does not appear, consult your System Administrator - the ttya device may need to be modified. Additional information on TIP can be found in the online man pages by typing **man tip**.

Some useful escape sequences to know when using TIP include (Note - it may be necessary to hit the **Enter** key or **CTRL-D** before entering these escape sequences.):

- **~?** win
- - help for TIP
- **~CTRL-D** - instructs the TIP command to terminate the connection and exit
- **~#** - sends a break to the remote system
- **~s script** - starts recording of transmissions made by the remote system

  Recordings are made in the default **tip.record** file in the user's current directory

- **~s !script** - stops recording of transmissions made by the remote system

Note - If a terminal emulator other than TIP is used, it must be configured for 9600 baud, eight bits per character, one stop bit, and no parity.

5.12 Booting the PowerPC 401 on the EVB

When the connectors have been installed and power is applied to the 401 EVB, pressing the Reset switch causes the 401 and the communications controllers to reset. After the ROM monitor initializes the 401 EVB, the monitor menu is displayed if a properly configured terminal (or terminal emulator) is attached to serial port 1 (J6) of the EVB. Details of ROM Monitor operation are provided in a later chapter.

6

401 EVB Hardware

Features of the 401 EVB include:

1) Memory
    - Contiguous addressing
    - DRAM, two SIMM sockets, up to 64MB
        - 4-1-1-1 accesses at 25 MHz bus speed
    - Two 512KB flash
    - SRAM, 512KB, socketed
        - 2-1-1-1 accesses at 25 MHz bus speed
        - Supports development of software for clock-stopped states
2) Battery-backed real-time clock with 8KB NVRAM
3) Two 16550-type serial ports
4) Ethernet
    - 10Base2 and 10BaseT
5) PS/2 parallel port
6) LCD - 2 lines of 16 characters
7) Processor-dependent interface
    - For logic analyzer or for processor-bus peripheral
8) Squall processor-independent expansion interface
    - For portable application-specific hardware support
9) Can function as PCI adapter card (I2O development)
10) Can function as PCI host to any standard PCI adapter card

6.1 401GF Overview

The PowerPC 401GF 32-bit RISC embedded controller offers high performance and functional integration with low power consumption. The PowerPC 401GF RISC CPU executes at sustained speeds approaching one cycle per instruction. On-chip caches reduce chip count and design complexity in systems, while improving system throughput. Figure 6-1 illustrates the logical organization of the PowerPC 401GF.

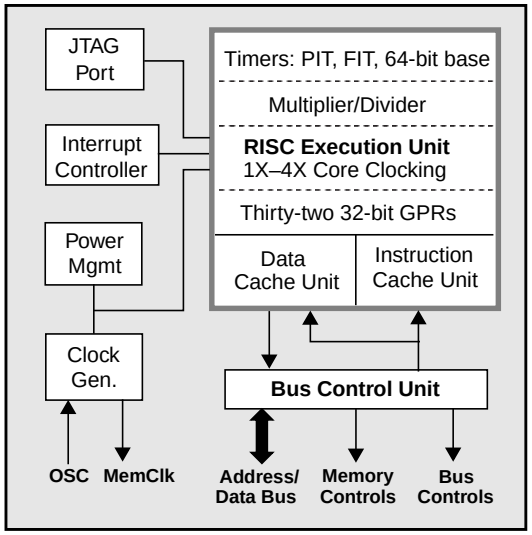


Figure 6-1. PowerPC 401GF Block Diagram

Features of the PowerPC 401GF include:

- PowerPC RISC fixed-point CPU and PowerPC User Instruction Set Architecture
  - Thirty-two 32-bit general purpose registers
  - Branch prediction
  - Single-cycle execution for most instructions
  - Hardware multiplier and divider for faster integer arithmetic
  - Enhanced string and multiple-word handling
- Interfaces to memory and peripherals
  - 32-bit data bus
  - Addressing for 4096MB of external memory and MMIO
  - Memory is pre-cache (cache tags are physical addresses)

    - Support for a wide range of memory timing parameters
    - Support for direct connection of byte, halfword, and fullword devices
- Storage attribute control
    - WIMGE (write-back/write thru, inhibited, guarded, Endian) storage attribute control for thirty-two 128MB regions
    - True Little-Endian operation and PowerPC Endian modes
- Separate instruction cache and write-back/write-thru data cache, both two-way set-associative
- Minimized interrupt latency
- Advanced power management
- Individually programmable on-chip interfaces for:
    - Bus control regions
    - External interrupts
- Flexible interface to external bus masters

6.2 Peripheral Components

1) Two Serial Ports and one PS/2 Parallel Port

   National PC16553DV.

2) Clock/calendar and Non-volatile RAM

   Dallas DS1643L-120.

   This chip is an 8Kx8 non-volatile RAM (by virtue of built-in lithium battery), with a clock/calendar replacing the top 8 bytes of the memory. It is mounted using a surface-mounted 52-pin PLCC socket.

3) Ethernet Controller

   National DP83902AV.

   This controller is used with 8Kx8 packet SRAM, National DP8392 coaxial transceiver interface, and both 10Base2 and 10BaseT connectors.

4) PCI

   PLX Technology PCI9060-3.

5) DRAM

The board is equipped with two 72-pin SIMM sockets, for 5 volt 70 nsec fast-page-mode DRAM with 130nsec read/write cycle time. The DRAM control is interleaved, therefore both sockets must be populated. Board logic accepts up to 64MB of DRAM. SIMMs may be single-sided or double-sided. The following restrictions on memory usage apply:

- Both SIMM sockets must be identically populated (both single-sided or both double-sided, and both the same memory size).
- Board jumpers specify the SIMMs. See Table 5-14 for jumper programming of SIMM size.

6) SRAM

Four 128Kx8 modules of 12 nsec SRAM in 32-pin 400-mil SOJ packages, socketed.

7) Flash

Two 512Kx8 devices, AMD 29F040, socketed.

8) Interrupt Controllers

Two 82C59 controllers, cascaded.

9) Power-On-Reset

Dallas DS1233.

6.3 Mechanical Specifications

The card outline (5.2x12.283 inch) is one inch taller than the standard full-sized PCI card (5V, 32 bit). The card will fit in PCs which can tolerate the extra height by removal of the cover. User access to the top edge of the card is assumed, since that edge mounts serial port and parallel port connectors and the expansion interface.

The card can be used in two ways:

1) As a PCI adapter card, assumed to be installed in a PC.

2) As a stand-alone development card, lying horizonal on a bench. The card will accept nylon standoffs to serve as "feet" for the card.

6.4 Displays

Table 6-1. Status LED Display

Location	Type	Description
CR1	LED	Ethernet Tx or Rx activity (green).
CR2	LED	Ethernet collision detected (yellow).
CR3	LED	Ethernet twisted-pair good link (green); not operational for coax.
CR4	LED	+5V present (green).
J31	LCD	2 line x 16 character intelligent LCD display with LED backlighting. FEMA CM162B-SGT1LY.

6.5 Switches

There are two switches on the 401 EVB. The Reset switch (SW1) on the board is a momentary SPST (Single Pole Single Throw) switch that generates a board hardware reset. A hardware reset simultaneously resets the 401 processor, the National DP83902AV Ethernet controller and the National PC16553 serial communications controller.The CRIT switch (SW2) on the board is a momentary SPST switch that generates a critical interrupt on the 401 processor chip. The ROM Monitor supports using the critical interrupt as a mechanism for suspending the execution of an application. When debug is not used, the ROM Monitor simply passes the critical interrupt on to the application's first level interrupt handler.

Table 6-2. 401 EVB Switches

Location	Description
SW1	Reset.
SW2	Manual critical interrupt (PCI SERR also generates critical interrupt).

6.6 Power Supply

The board is designed to run from a single +5V supply. Other needed voltages (+3.3V for the 401GF chip; RS232 voltages) are derived on the board. Either a standard PC planar power connector or the PCI edge connector will power the board; both sources of power will never be simultaneously connected.

When the board is hosting a Squall module, +12V and -12V may be required. When the board is configured as a PCI adapter card, these voltages are supplied from the PCI edge connector. When the board is configured for stand-alone operation, these voltages are supplied via separate connectors.

When the board is configured for stand-alone operation and is hosting PCI card, +3.3V, +12V, and -12V may be required. +12V and -12V are supplied via separate connectors. +3.3V may be supplied from the on-board regulator or via a separate connector, depending on the current requirements.

6.7 Initialization

Information regarding the reset and initialization of the 401GF controller can be found in the 401GF User's Manual. Minimum board initialization consists of the following:

SGR ← 0	optional, to improve execution speed
DCWR ← 0	before first store, to prevent alignment exception
SLER ← 0x0BC00BC0	set endian regions (little-endian for PCI and Squall, regions 4,6-9,20,22-25; all other regions big-endian)
IOCR ← 0x40000000	set active levels for interrupts (critical low, non-critical high)
BRCR0 ← 0x1A	set DRAM memory region (32-bit, 16-burst, target-first,1-hold)
BRCR1 ← 0x1A	set SRAM memory region (32-bit, 16-burst, target-first,1-hold)
BRCR2 ← 0x1A	set PCI Registers memory region (32-bit, 16-burst, target-first,1-hold)
BRCR3 ← 0x1A	set Local-to-PCI memory region (32-bit, 16-burst, target-first,1-hold)
BRCR4 ← 0x1A	set Squall memory region (32-bit, 16-burst, target-first,1-hold)
BRCR7 ← 0x18	set Peripherals memory region (8-bit, 16-burst, target-first,1-hold)

Example code which accomplishes this follows:

```
FFFFFF00    addis    r3,0,0
FFFFFF04    ori      r3,r3,0
FFFFFF08    mtspr    SGR,r3
FFFFFF0C    mtspr    DCWR,r3
FFFFFF10    addis    r3,0,0x0BC0
FFFFFF14    ori      r3,r3,0x0BC0
FFFFFF18    mtspr    SLER,r3
FFFFFF1C    addis    r3,0,0x4000
FFFFFF20    ori      r3,r3,0
FFFFFF24    mtdcr    IOCR,r3
FFFFFF28    addis    r3,0,0
```

```
FFFFFF2C    ori     r3,r3,0x001A
FFFFFF30    mtdcr   BRCR0,r3
FFFFFF34    mtdcr   BRCR1,r3
FFFFFF38    mtdcr   BRCR2,r3
FFFFFF3C    mtdcr   BRCR3,r3
FFFFFF40    mtdcr   BRCR4,r3
FFFFFF44    addis   r3,0,0
FFFFFF48    ori     r3,r3,0x0018
FFFFFF4C    mtdcr   BRCR7,r3
FFFFFF50    nop
FFFFFF54    nop
FFFFFF58    nop
FFFFFF5C    nop
FFFFFF60    nop
FFFFFF64    nop
FFFFFF68    b       $

FFFFFFFC    b       $-0xFC  #branch to FFFFFF00
                            #entry point of init code
```

6.8 Endian Programming

PCI and Squall memory regions are most naturally little-endian. Those regions are:

4 and 20	PCI Registers	0x20000000-0x200001FF and 0xA0000000-0xA00001FF
6 and 22	Local-toPCI I/O	0x30000000-0x37FFFFFF and 0xB0000000-0xB7FFFFFF
7 and 23	Local-toPCI Memory	0x38000000-0x3FFFFFFF and 0xB8000000-0xBFFFFFFF
8,9 and 24,25	Squall Interface	0x40000000-0x4FFFFFFF and 0xC0000000-0xCFFFFFFF

All other regions are typically programmed as big-endian.

6.9 Utility Port

Miscellaneous board control functions are handled by the Utility Port located at address 0x7E000200. This port controls critical interrupts, slow-mode refresh, and Squall serial EEPROM access. Bits in this port are write-only except bits 7 and 4, which are read/write. This port resets to all bits low.

Table 6-3. Utility Port Bit Assignments

Bit	Description	Access
7 (MSb)	Squall EEPROM serial data	Read/Write
6	Squall EEPROM serial data enable 0=hiZ 1=enable	Write Only
5	N/C	Write Only
4	Squall EEPROM serial clock	Read/Write
3	Critical Interrupt Acknowledge 0=crit irpt enabled 1=remove crit irpt	Write Only
2 1	N/C N/C	
0 (LSb)	Slow Mode 0=refresh rate for 25 MHz 1=refresh rate for 5MHz	Write Only

6.10 Ethernet Remote DMA Ports

According to the data sheet of the 83902 Ethernet chip, the PRQ signal is a handshake control between the Ethernet chip and its host processor. During DMA reads (Ethernet to 401), the 83902 places PRQ high to indicate that data is available. During DMA writes (401 to Ethernet), the 83902 places PRQ high to indicate that it is prepared to accept data.

Therefore, this board has one address (0x74000010) to read the status of PRQ. It appears as the high-order bit of the byte. This port address is read-only.

When a read of the PRQ Status port finds PRQ high, Ethernet Remote DMA data may be read or written to 0x74000011, which is an exchange port between the 401 and the packet SRAM which is managed by the 83902.

Table 6-4. Ethernet DMA PRQ Status

Bit	Description	Access
7	Ethernet PRQ	Read Only
6:0	undefined	

Table 6-5. Ethernet Remote DMA Port

Bit	Description	Access
7:0	Ethernet data	Read/Write

6.11 401 EVB Memory Map

The memory map of the 401 EVB is as follows:

PowerPC 401GF Board Memory Map	
FFF8 0000 -> FFFF FFFF FFF0 0000 -> FFF7 FFFF F000 0000 (BRCR 7) 8-bit	Boot Flash, 512K PCI Expansion Flash, 512K
  E000 0000 (BRCR 6)	(unused)
  D000 0000 (BRCR 5)	(unused)
C000 0000 -> CFFF FFFF C000 0000 (BRCR 4) 32-bit	Squall, 256M
B800 0000 -> BFFF FFFF B000 0000 -> B7FF FFFF B000 0000 (BRCR 3) 32-bit	Local-to-PCI Memory, 128M Local-to-PCI I/O, 128M
  A000 0000 (BRCR 2) 32-bit	
  9000 0000 (BRCR 1) 32-bit	
  8000 0000 (BRCR 0) 32-bit	

PowerPC 401GF Board Memory Map (Continued)	
7E01 0000 -> 7E01 1FFF	RTC/NVRAM, 8K
7E00 0480 -> 7E00 0481	Interrupt Controller 2, slave (82C59)
7E00 0402	Interrupt Ack from both Interrupt Controllers
7E00 0400 -> 7E00 0401	Interrupt Controller 1, master (82C59)
7E00 0300 -> 7E00 0301	LCD (CM162B)
7E00 0200	Utility port (controls critical interrupt, slow-mode refresh, and Squall serial EEPROM access)
7E00 0100 -> 7E00 0103	PS/2 Parallel Port (16553)
7E00 0080 -> 7E00 0087	Serial Port (16553)
7E00 0000 -> 7E00 0007	Serial Port (16553)
7400 0011	Ethernet Remote DMA, data
7400 0010	Ethernet PRQ Status, read-only
7400 0000 -> 7400 000F	Ethernet (83902)
7000 0000 (BRCR 7) 8-bit	
	(unused)
6000 0000 (BRCR 6)	
	(unused)
5000 0000 (BRCR 5)	
4000 0000 (BRCR 4) 32-bit	
3000 0000 (BRCR 3) 32-bit	
2000 0000 -> 2000 01FF	PCI Registers (not all addresses used)
2000 0000 (BRCR 2) 32-bit	
1000 0000 -> 1007 FFFF	SRAM, 512K
1000 0000 (BRCR 1) 32-bit	

PowerPC 401GF Board Memory Map (Continued)	
0000 0000 -> 03FF FFFF	DRAM, 64M max
0000 0000 (BRCR 0) 32-bit	

Notes:	
1)	The hi-order address bit is not decoded on this board. Therefore, everything in the memory map actually appears twice, once with the bit = 1, and once with the bit = 0.
2)	DRAM is aliased at addresses 0400 0000 -> 07FF FFFF. Per (1), DRAM also appears at 8400 0000 -> 87FF FFFF.
3)	There are no aliased addresses other than those described in (1) and (2).

6.12 Squall Expansion Interface

The Squall interface is an open interface for expansion cards. The interface is defined by Cyclone Microsystems. Each type of Squall module contains a unique software-readable registration number. Designers of new Squall modules should request a number from:

Cyclone Microsystems
25 Science Park
New Haven, CT 06511
phone: 203-786-5536
fax: 203-786-5025
email: info@cyclone.com

Every Squall module contains a 24C08 serial EEPROM (2K bytes) which is used to identify the type and revision of the module, and to store module-dependent system parameters. The first 10 bytes are used identically on all Squall modules. The remaining memory is assigned and documented by the module designer. The general Squall memory map is defined as follows:

Table 6-6. Squall Module Memory Map

Location	Description
0-3	Region Configuration Word. Stored little-endian.
4	Interrupt Detection Mode. SQIRQ0 — bit 7 (LSb). SQIRQ1 — bit 6. A bit value of 0 indicates level-sensitive; a bit value of 1 indicates edge-sensitive.
5-6	Reserved
7-8	Module Version. Stored in ASCII. Value should be assigned by Cyclone Microsystems.
9	Module Revision Level. Stored in binary. Value is assigned by the module designer.
0x00A-0x7FF	Module-dependent data.

The EEPROM is read and written serially using the Utility Port at U23. Bytes are read most significant bit first.

6.13 Non-Critical Interrupts

The non-critical interrupt input to the PowerPC 401GF should be programmed for an active-high level (IOCR[IL] = 1).

Non-critical interrupts are handled by a pair of cascaded 82C59 interrupt controllers. All inputs to the slave controller are level-sensitive, and that controller should be programmed accordingly. Inputs to the master controller may be either edge-sensitive or level-sensitive. This is determined by reading the Interrupt Detection Mode of the Squall module; if no Squall module is installed, then the master controller should be programmed as level sensitive.

Table 6-7. Non-critical Interrupts, Controller 1

(master controller, programmed edge or level sensitive based on reading Interrupt Detection Mode from Squall EEPROM)			
Default Priority	**Controller Int Level**	**Source**	**Comments**
(1-8)	0	Interrupts from Controller 2	(Hi-priority unless altered by Specific Rotation in OCW2 of Controller 1.)
9	1	Squall -SQIRQ0	(May be edge or level sensitive.)
10	2	Squall -SQIRQ1	(May be edge or level sensitive.)

Table 6-8. Non-critical Interrupts, Controller 2

(slave controller, programmed level-sensitive)			
Default Priority	**Controller Int Level**	**Source**	**Comments**
1	0	PCI LINT	
2	1	PCI LSERR	
3	2	PCI BREQ	
4	3	PCI INT	
5	4	Ethernet	
6	5	Serial Port 1	
7	6	Serial Port 2	
8	7	Parallel Port	

6.14 Critical Interrupts

The critical interrupt input to the 401GF should be programmed for an active-low level (IOCR[CIL] = 0).

Critical interrupt inputs to the PowerPC 401GF are latched in board hardware. The latch is cleared by writing a 1 to the Utility Port INT_REMOVE bit, then writing a 0 to the same bit. The critical interrupt will be removed from the PowerPC 401GF immediately upon writing the 1 to INT_REMOVE. Critical interrupt cannot recur until both INT_REMOVE and the original interrupt source have been cleared.

Table 6-9. Critical Interrupts

Source	Comments
Critical Interrupt switch	
PCI SERR	(Duration is one PCI bus clock cycle.)

6.15 Network Address of the Ethernet Controller

The EVB Ethernet controller, a National DP83902AV, has been assigned a unique six-byte network address. This address, also known as the media access control or MAC address, may need to be known by customers using the EVB to develop their own ROM versions.

The easiest way to obtain its value is to hook up a terminal (or terminal emulator) to the EVB serial port 1 (as explained in the previous chapter) and bring up the ROM Monitor. After selecting option 7 to display the configuration, the controller's network address is displayed in the Ethernet boot source's *hwaddr* field as twelve hex characters (six bytes).

The ROM Monitor returns the MAC address as part of the board_cfg_data structure when a call to its get_board_config() function is made. Sample code showing how this is done can be found in the usr_samp.c file in the OS Open samples directory.

Another way to obtain the address, is to search the Vital Product Data (VPD) area in ROM where the network address is stored. The VPD fields consist of ASCII strings identifying the type of field, a length byte specifying the length of the associated data, and the data itself. The VPD begins at address 0xFFFF FE00 and is marked by field "*VPD" with 0 bytes of associated data. The network address is marked by "*NA" with six bytes of associated data (the network address). Finally, the end of the VPD is marked with "*END". To extract the network address, a program would typically start at 0xFFFF FE00, scan for "*NA", verify the next byte is 0x6, and treat the next six bytes as the network address.

7

401 EVB ROM Monitor

This chapter describes the 401 EVB ROM Monitor program. This ROM resident program provides chip (and board level) initialization and a user interface menu that supports board diagnostics, program downloads, and debug.

7.1 ROM Monitor Source Code

The ROM Monitor source code is provided for ROM development purposes. This code is separate from the sample applications described in Chapter 8. The code is loosely organized by function in the following subdirectories and files within the **/usr/osopen/m401_evb/openbios** directory (**\osopen\m401_evb\openbios** for PC users).

- Makefile — Top level makefile to create ROM monitor image (RS/6000 & SUN)
- makefile.mak — Top level makefile to create ROM monitor image (PC)
- devTab.c — Handles boot device definitions
- include/ — C include files
- m4/ — assembler preprocessor include files
- ppcLib/ — C callable functions to access PowerPC special instructions
- enetLib/ — Ethernet chip specific code
- ioLib/ — I/O helper functions
- miscLib/ — Miscellaneous routines used for ROM monitor
- s1Lib/ — Serial Port interface routines
- s1ldLib/ — Code to support S1 serial port downloads
- dbLib/ — Ptrace debug interface routines
- entry.s — Processor and C environment initialization
- lib/ — Repository for intermediate libraries
- netLib/ — IP and UDP processing functions
- slipLib/ — SLIP implementation
- align_h.s — Alignment handling code
- mapfile1 — Mapfile to specify ROM Monitor linkage directives
- bios_***.map — Load map of the ROM Monitor version *** shipped with the EVB
- flash/ — Code to support re-programming the flash memory
- lcdLib — LCD access functions

7.2 Communications Features

The 401 EVB ROM Monitor runs as part of the boot code in the flash memory on the board. The monitor communicates with an asynchronous terminal (or terminal emulator) attached to serial port 1 (SP1) on the EVB, through which the user accesses the monitor menu. The 401 EVB can download applications and communicate with the host debugger through serial port 2 (SP2) or the Ethernet adapter, depending on which devices are enabled. Communications between SP2 and the host use the Serial Link Internet Protocol (SLIP), while Ethernet communications use the Internet Protocol (IP) over standard Ethernet. The 401 EVB also supports the downloading of programs via serial port 1 (SP1). To use this feature, a VT100 terminal emulator that supports binary file transfers (such as kermit) must be used on the host system.

7.3 Bootp and tftp Configuration to support ROM Monitor Loads

Both the debugger and the ROM Monitor can be used to load applications onto the board. Details on how to use the debugger can be found in the *RISCWatch* Debugger *User's Guide*. To use the facilities of the ROM Monitor for downloading applications to the evaluation board, the host workstation must be configured to support the **bootp** protocol and **tftp** daemons. The configuration consists of two parts. The **bootptab** file on the host must be customized to match system requirements, and the **bootp** and **tftp** daemons (or servers) must be made available.

7.3.1 RS/6000 bootp and tftp configuration

To modify the **/etc/bootptab** file, you need to log in as **root** or the superuser (**su**). Entries describing the evaluation board to the host workstation must be added to this file. Complete details describing the bootptab file format are available in the *AIX Command Reference* under "bootpd". File entries suitable for our purposes are shown below.

```
slipc:hd=/usr/osopen/m401_evb/samples:bf=boot.img:bs:ip=8.1.1.5:sm=255.
255.255.255
enetc:ht=ethernet:hd=/usr/osopen/m401_evb/samples:bf=boot.img:bs:ip=7.1
.1.5:sm=255.255.255.255:ha=xxxxxxxxxxxx
```

Each of the entries, slipc and enetc, should be entered on a single line. The value of the Ethernet hardware address field in the enetc entry, ha=xxxxxxxxxxxx, should match the twelve character hardware address listed for the Ethernet Boot Source on the ROM Monitor menu.

Both connections use the file **/usr/osopen/m401_evb/samples/boot.img** as the source for the application image to be downloaded onto the board. Be sure that the **ht=ethernet** keyword is used for the Ethernet connection entry and that the IP addresses are those of the evaluation board. Note that the IP address in the slipc entry must match that of the IP address assigned to the board during serial port set-up. Since a board IP address was not required for Ethernet set-up, the IP address used in the enetc entry defines the IP address of the board for the Ethernet connection. If the suggested bootptab entries are used, 7.1.1.5 would be the board's Ethernet IP address. Take note of the board's IP addresses, since they must be made known to the ROM Monitor.

To start the **bootp** and **tftp** daemons on systems running AIX 3, do the following:

- log in as **root** or the superuser (**su**)
- enter **smit**
- select **Diskless Workstation Management and Installation**
- select **Start Daemons on Server**
- select **Start BOOTP Daemon**
- select **Do** or hit **Enter**

  Upon successful completion, **bootp** configuration is complete. Continue for **tftp**:

- select **Done** or hit **PF3**
- select **Cancel** or hit **PF3** to return to the **Start Daemons on Server** screen
- select **Start TFTP Daemon**
- select **List**

  If "**tftp udp**" is not on the list, **tftp** has already been started for the workstation. The configuration steps are complete. Select **Exit** to leave **smit**.

- select "**tftp udp**"
- select **Do** or hit **Enter**
- You should be at the **Add an inetd Subserver** screen. The defaults listed are acceptable.
- select **Do** or hit **Enter**

  Upon successful completion, **tftp** configuration is complete. Select **Exit** to leave **smit**

To start the **bootp** and **tftp** daemons on systems running AIX 4, do the following:

- log in as **root** or the superuser (**su**)
- enter **smit**
- select **Processes and Subsystems**
- select **Subservers**
- select **Start a Subserver**
- select **bootps**
- select **OK**

  Upon successful completion, bootp configuration is complete. Select **Done** and continue for **tftp.**

- select **Start a Subserver**
- select **tftp**
- select **OK**
- select **Done**

  Upon successful completion, **tftp** configuration is complete. Select **Exit** to leave **smit**

7.3.2 PC bootp and tftp configuration

Not all TCP/IP packages include the bootpd and tftpd servers required for ROM Monitor downloads. For this reason both the bootpd and tftpd servers have been included in the EVB software package under the \osopen\bin directory. These servers can be installed and used in conjunction with Windows Socket compliant TCP/IP packages such as Trumpet Winsock and those that come with Windows 95 and Windows NT.

Since TCP/IP packages vary greatly, this section should be used only as a **guideline** for bootp and tftp set-up. Users should consult their TCP/IP documentation for specific details.

Configuration consists of two parts. The **bootptab** and **services** files on the host must be customized to match system requirements, and the bootpd and tftpd servers must be made available. If you choose to use the bootpd and tftpd servers provided with this package, you will need to modify your **autoexec.bat** file to specify the location of the bootptab and services files. This is accomplished by adding a line that sets up an ETC constant to the directory where the bootptab and services files are located (ie. SET ETC=C:\TRUMPET for Windows 3.1/Windows 95 Trumpet users, ETC=C:\WINDOWS for Windows 95 users, ETC=C:\WINNT35\system32\drivers\etc for Windows NT 3.51).

A sample bootptab file, **\osopen\PLATFORM\samples\bootptab.sam**, is included with the EVB software. This file can be copied to the ETC directory set in the autoexec.bat file and modified appropriately. Note that the bootptab file in the ETC directory must be named **bootptab** with no file extention. Entries describing the evaluation board to the host PC must be added to the bootptab file.

When creating or modifying the bootptab file, the following rules apply.

- blank lines and lines beginning with “#” are ignored.
- each entry must be entered on a single line.
- each entry must start with a hostname followed by the legends (see the sample bootptab file for legend descriptions).
- use “:” to separate each legend and leave no spaces between legends.
- user must supply the host ip address via the “ip” legend.
- if the “hd” (home directory) & “bf” (bootfile) legends are not provided for a particular entry, the first defined “hd” and “bf” legends in the bootptab file will be taken as default.

File entries similar to those below would be suitable.

```
slipc:hd=\osopen\PLATFORM\samples:bf=boot.img:bs:ip=8.1.1.5:sm=255.255.
255.255
enetc:ht=ethernet:hd=\osopen\PLATFORM\samples:bf=boot.img:bs:ip=7.1.1.5
:sm=255.255.255.255:ha=xxxxxxxxxxxx
```

Each of the entries, slipc and enetc, should be entered on a **single** line. The value of the Ethernet hardware address field in the enetc entry, ha=xxxxxxxxxxxx, should match the twelve character hardware address listed for the Ethernet Boot Source on the ROM Monitor menu.

Both connections use the file **\osopen\PLATFORM\samples\boot.img** as the source for the application image to be downloaded onto the board. Be sure that the **ht=ethernet** keyword is used for the Ethernet connection entry and that the IP addresses are those of the evaluation board. Note that the IP address in the slipc entry must match that of the IP address assigned to the board during serial port set-up. Since a board IP address was not required for Ethernet set-up, the IP address used in the enetc entry defines the IP address of the board for the Ethernet connection. If the suggested bootptab entries are used, 7.1.1.5 would be the board's Ethernet IP address. Take note of the board's IP addresses, since they must be made known to the ROM Monitor.

The **services** file (no file extention) must also exist in the ETC directory set in the autoexec.bat file. It must be updated with the port and protocol information for the bootpd and tftpd servers. To use the servers provided with this package, the following entries must be included in the services file.

```
bootps      67/UDP
bootpc      68/UDP
tftp        69/UDP
```

For the update to take effect, TCP/IP needs to be re-started. This may require a re-boot of the system and/or a restart of the TCP/IP package. After that, the bootpd and tftpd servers are ready for use.

7.3.2.1 Automatic startup for Windows 3.1 and Windows NT 3.51

Users may find it convenient to have the bootpd and tftpd servers brought up automatically when entering Windows. To do this for Windows 3.1, the bootpd and tftpd servers should be added to your Windows environment Startup window using the following procedure.

- With Windows running, select the Program Manager and open the Startup window.
- Using the File pulldown menu on the Program Manager, select New to bring up a New Program Object window.
- From the New Program Object window, select Program Item and OK to open the Program Item Properties window. The Program Item Properties window requires that you provide Description, Command Line and Working Directory values. The following example shows one possible configuration.

```
Description:  BOOTPD
Command Line: BOOTPD -C D -H 7.1.1.4
Working Directory: D:\OSOPEN\BIN
```

In the above example, the command line specifies how to invoke the bootpd server, and the working directory specifies where to find the bootpd server program (bootpd.exe). The -C parameter is used to specify a drive letter that is used in conjunction with bootptab file entries. Because the colon is used as a delimiter in bootptab file entries, the -C parameter is used as a mechanism by the bootpd server to concatonate a drive letter to the beginning of the hd: field. If the -C option is not specified, the current drive will be used as a default. The -H parameter is used to specify the Ethernet or slip IP address of the host PC (set during host configuration) to the bootpd server.

Use the same procedure to set up the tftpd server. In this case, the Program Item Properties window entries will describe information used for the tftpd server. The following example shows a possible configuration.

Description: TFTPD

Command Line: TFTPD

Working Directory: D:\OSOPEN\BIN

If you do not wish to have the bootpd and tftpd servers run automatically upon entering Windows, they can be run individually from the Windows Program Manager, File, Run menu. Note that TCP/IP must be up and running before the servers can be run.

7.3.2.1 Automatic startup for Windows 95

You may choose to run "BOOTPD.EXE" and "TFTPD.EXE" automatically every time that WIndows 95 is started or you can run these programs only when needed. To make these program run automatically every time WIndows 95 is started perform the following steps.

- Select 'Start' from the Windows 95 task bar.
- Select 'Settings'.
- Select 'Taskbar'.
- Select 'Start Menu Programs'.
- Select 'Add...'.
- In the command line field enter the following:
  BOOTPD -c C -h 7.1.1.4
  (Where "C" is the driver letter containing the boot image and "7.1.1.4" is host IP address)
- Select ‘Next’.
- In the ‘Select Program Folder’ window, select the ‘Programs/Startup’ folder.
- Select ‘Next’.
- Select ‘Finished’.
- To start "TFTP" follow the above steps, but enter the following in the command line field:
  TFTPD.

The BOOTP and TFTP demons will be started automatically upon the next restart of Windows 95.

7.3.3 SUN bootp and tftp configuration

The Solaris and SunOS operating systems both provide a tftpd server but do not provide a bootpd server. For this reason a bootpd server has been included in the EVB software package under the /usr/osopen/bin directory.

A sample bootptab file, **/usr/osopen/PLATFORM/samples/bootptab.sam**, is included with the EVB software. This file should be copied to the **/etc** directory and renamed **bootptab** if a bootptab file does not already exist. You will need to log in as root or the superuser (su) to update or add files in the /etc directory. Entries describing the evaluation board to the host PC must be added to the bootptab file.

When creating or modifying the bootptab file, the following rules apply.

- blank lines and lines beginning with “#” are ignored.
- each entry must be entered on a single line.
- each entry must start with a hostname followed by the legends (see the sample bootptab file for legned descriptions).
- use “:” to separate each legend and leave no spaces between legends.
- user must supply the host ip address via the “ip” legend.
- if the “hd” (home directory) & “bf” (bootfile) legends are not provided for a particular entry, the first defined “hd” and “bf” legends in the bootptab file will be taken as default .

File entries similar to those below would be suitable.

```
slipc:hd=/usr//osopen/PLATFORM/samples:bf=boot.img:bs:ip=8.1.1.5:sm=255
.255.255.255
enetc:ht=ethernet:hd=/usr/osopen/PLATFORM/samples:bf=boot.img:bs:ip=7.1
.1.5:sm=255.255.255.255:ha=xxxxxxxxxxxx
```

Each of the entries, slipc and enetc, should be entered on a **single** line. The value of the Ethernet hardware address field in the enetc entry, ha=xxxxxxxxxxxx, should match the twelve character hardware address listed for the Ethernet Boot Source on the ROM Monitor menu.

Both connections use the file /usr/**osopen/PLATFORM/samples/boot.img** as the source for the application image to be downloaded onto the board. Be sure that the **ht=ethernet** keyword is used for the Ethernet connection entry and that the IP addresses are those of the evaluation board. Note that the IP address in the slipc entry must match that of the IP address assigned to the board during serial port set-up. Since a board IP address was not required for Ethernet set-up, the IP address used in the enetc entry defines the IP address of the board for the Ethernet connection. If the suggested bootptab entries are used, 7.1.1.5 would be the board’s Ethernet IP address. Take note of the board’s IP addresses, since they must be made known to the ROM Monitor.

To start the bootpd and tftpd servers:

- log in as **root** or the superuser (**su**)
- ensure that the following entries are included in the **/etc/services** file.
    - `bootps 67/udp`
    - `bootpc 68/udp`
    - `tftp 69/udp`
- ensure that the tftp entry in the **/etc/inetd.conf** file is uncommented and modify as follows.

    - `tftp dgram udp wait root /usr/etc/in.tftpd in.tftpd -s /`
- add an entry for the bootpd server in **/etc/inetd.conf** as follows.
    - bootps dgram udp wait root /usr/osopen/bin/bootpd bootpd -i
- reconfigure inetd for the updates made to the inetd.conf file. First find the process id for inetd .
    - `ps -ef | grep inetd` (Solaris)
    - `ps -auex | grep inetd` (SunOS)
- Then send a hangup signal to reconfigure inetd.
    - `kill -HUP <process id>`

Bootp and tftp configuration is complete.

7.4 Accessing the ROM Monitor

The ROM Monitor expects a real or emulated VT100 type ASCII display attached to serial port 1 with line protocol parameters of 9600 baud, eight bits per character, no parity, and one stop bit. Once the terminal connected to SP1 is configured properly, you can access the ROM Monitor menu options, use the ping test, and load an application onto the evaluation board.

The ROM Monitor also provides the interface to the RISCWatch debugger. This facility, along with the image download process, is accessed via an IP network connnection to the host workstation. Network configuration of the host was discussed earlier in the chapter on host configuration. The actual connection is either via SLIP (Serial Link Interface Protocol) running on serial port 2 at speeds up to 56K baud, or via standard Ethernet using the 10Base2 or 10BaseT connector on the evaluation board.

7.5 ROM Monitor Operation

The ROM Monitor requires a block of DRAM for its operation and makes some assumptions about applications loaded on the board. Some of these assumptions may be disregarded if you do not need the ROM Monitor to interface with a debugger or otherwise support communication between the host workstation and the EVB.

Applications wishing to coexist with the ROM Monitor must observe the following constraints.

- Do not alter the EVPR register.
- Provide exception vectors for application events as if the EVPR were set to 0x0000 0000. For example, an application's external interrupt handler should be located at 0x0000 0500. This is handled for you when using OS Open.
- Use storage addresses between 0x0000 A000 and the end of DRAM only, except for application vectors.

- Do not start applications lower than address 0x0000 A000.

Figure 7-1 shows the address map of the evaluation board under control of the ROM Monitor.The “folding” characteristics of the high order address bit are not shown.

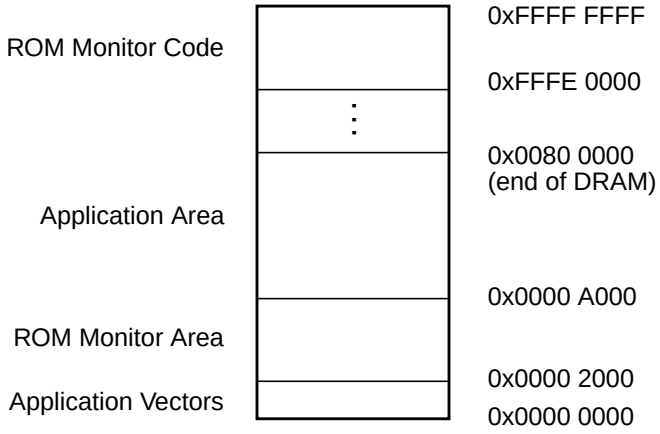


Figure 7-1. ROM Monitor Address Map

7.6 Monitor Selections and Submenus

At this point it is assumed that the host has been properly configured, all board connections have been made, power has been supplied, and the terminal emulator running on the host has been configured and started successfully. The main menu, shown below, is displayed after the 401 EVB has been reset and the ROM Monitor completes initialization. Note that some of the values you see, in particular the ROM Monitor version, the IP addresses, and the Ethernet controller’s hardware address, may differ with those shown below.

Each menu option is described separately in the following sections. “Local” in the context of the ROM Monitor IP addressing means the IP address assigned to the evaluation board, while “remote” means the IP address assigned to the host workstation. Using option 8 to save changes made to the configuration will allow the new values to persist beyond subsequent power-ons or resets. The ROM Monitor supports this by storing its configuration data in NVRAM.

7.6.1 Initial ROM Monitor Menu

The following menu is displayed after the board has been reset.

```
401GF 2.1 ROM Monitor (8/2/96)

  ------------- System Info -------------
  Processor speed   =  50 MHz
  Bus speed            =  25 MHz
  Amount of DRAM  =  8 MB
  -----------------------------------------
```

```
  --- Device Configuration ---
  Power-On Test Devices:
     000  Enabled    System Memory [RAM]
     001  Enabled    Ethernet  [ENET]
     004  Enabled    Serial Port 2 [S2]
  ----------------------------------------
  Boot Sources:
     001  Enabled   Ethernet  [ENET]
                local=7.1.1.5  remote=7.1.1.4  hwaddr=1000abcdef55
     004  Enabled   Serial Port 2 [S2]
                local=8.1.1.5  remote=8.1.1.4  hwaddr=ffffffffffff
     005  Enabled   Serial Port 1 [S1]
               Baud = 9600
    ----------------------------
    Debugger : Disabled
    ----------------------------
   1 - Enable/disable tests
   2 - Enable/disable boot devices
   3 - Change IP addresses
   4 - Ping test
   5 - Toggle ROM monitor debugger
   6 - Toggle automatic menu
   7 - Display configuration
   8 - Save changes to configuration
   9 - Set baud rate for s1 boot
   0 - Exit menu and continue
->
```

7.6.2 Selecting Power-On Tests

Option 1 in the main menu selects power-on tests. These tests are run when the menu exits and before the ROM loader begins the bootp processing.

```
 1 - Enable/disable tests
 2 - Enable/disable boot devices
 3 - Change IP addresses
 4 - Ping test
 5 - Toggle ROM monitor debugger
 6 - Toggle automatic menu
 7 - Display configuration
 8 - Save changes to configuration
 9 - Set baud rate for s1 boot
 0 - Exit menu and continue
->1
```

When option 1 is selected, the following submenu is displayed.

```
--- ENABLE AND DISABLE POWER-ON TESTS ---
 Power-On Test Devices:
    000   Enabled     System Memory [RAM]
    001   Enabled     Ethernet  [ENET]
    004   Enabled     Serial Port 2 [S2]
----------------------------
select device to change ->
```

Selecting a test toggles its testing status. For example, since the System Memory test is enabled in the above menu, selecting 0 at the prompt disables it.

```
select device to change ->0             [Selects system memory]
```

After the selection has been made, the new setting is displayed, followed by the main menu.

```
select device to change ->0
   [RAM] test is disabled               [Message describing change]

  --- Device Configuration ---
  Power-On Test Devices:
     000  Disabled   System Memory [RAM]
     001  Enabled    Ethernet  [ENET]
     004  Enabled    Serial Port 2 [S2]
  ----------------------------
  Boot Sources:
     001  Enabled   Ethernet  [ENET]
               local=7.1.1.5  remote=7.1.1.4  hwaddr=1000abcdef55
     004  Enabled   Serial Port 2 [S2]
               local=8.1.1.5  remote=8.1.1.4  hwaddr=ffffffffffff
     005  Enabled   Serial Port 1 [S1]
              Baud = 9600
  ----------------------------
```

```
    Debugger : Disabled
    ---------------------------
     1 - Enable/disable tests
     2 - Enable/disable boot devices
     3 - Change IP addresses
     4 - Ping test
     5 - Toggle ROM monitor debugger
     6 - Toggle automatic menu
     7 - Display configuration
     8 - Save changes to configuration
     9 - Set baud rate for s1 boot
     0 - Exit menu and continue
  ->
```

Remember to use Option 8 to save any configuration changes that you may have made. If the changes are not saved, they will be lost upon an exit from the menu or upon a board reset.

7.6.3 Selecting Boot Devices

Option 2 in the main menu enables and disables boot devices.

```
 1 - Enable/disable tests
 2 - Enable/disable boot devices
 3 - Change IP addresses
 4 - Ping test
 5 - Toggle ROM monitor debugger
 6 - Toggle automatic menu
 7 - Display configuration
 8 - Save changes to configuration
 9 - Set baud rate for s1 boot
 0 - Exit menu and continue
->2
```

When option 2 is selected, the following submenu is displayed.

```
--- ENABLE AND DISABLE BOOT DEVICES ---
 Boot Sources:
     001   Enabled     Ethernet  [ENET]
                                 local=7.1.1.5  remote=7.1.1.4
hwaddr=1000abcdef55
     004   Enabled     Serial Port 2 [S2]
                                 local=8.1.1.5  remote=8.1.1.4
hwaddr=ffffffffffff
     005   Enabled     Serial Port 1 [S1]
                 Baud = 9600
 ----------------------------
select device to change ->
```

Selecting a device toggles its boot status. Selecting 4, for example, would disable Serial Port 2 as a boot device.

```
select device to change ->4              [Selects serial port]
```

After the selection has been made, the new setting is displayed, followed by the main menu.

```
select device to change ->4
   [S2] boot is disabled                [Message describing change]

  --- Device Configuration ---
  Power-On Test Devices:
     000  Disabled   System Memory [RAM]
     001  Enabled    Ethernet  [ENET]
     004  Enabled    Serial Port 2 [S2]
  ----------------------------
  Boot Sources:
     001  Enabled   Ethernet  [ENET]
```

```
               local=7.1.1.5  remote=7.1.1.4  hwaddr=1000abcdef55
      004  Disabled  Serial Port 2 [S2]
               local=8.1.1.5  remote=8.1.1.4  hwaddr=ffffffffffff
      005  Enabled   Serial Port 1 [S1]
               Baud = 9600
    ----------------------------
    Debugger : Disabled
    ----------------------------
     1 - Enable/disable tests
     2 - Enable/disable boot devices
     3 - Change IP addresses
     4 - Ping test
     5 - Toggle ROM monitor debugger
     6 - Toggle automatic menu
     7 - Display configuration
     8 - Save changes to configuration
     9 - Set baud rate for s1 boot
     0 - Exit menu and continue
  ->
```

When the user selects option 0 and exits from the monitor menu, the monitor attempts a boot of the application image on the host using the enabled boot sources in the order they are listed. In the above example, a boot would be attempted over Ethernet since it is the first boot source enabled. If more than one boot source is enabled, an attempt to boot over the first enabled device will be made. If that attempt fails, a boot over the next enabled device is attempted.

7.6.4 Changing IP Addresses

Option 3 in the main menu allows users to change the IP addresses for the EVB and the host workstation. These addresses are used for bootp processing, debugger communications, and in the host connectivity “ping” test. **Note** - the local IP address is that of the board and the remote IP address is that of the host workstation. The IP addresses must match those set during host configuration.

```
 1 - Enable/disable tests
 2 - Enable/disable boot devices
 3 - Change IP addresses
 4 - Ping test
 5 - Toggle ROM monitor debugger
 6 - Toggle automatic menu
 7 - Display configuration
 8 - Save changes to configuration
 9 - Set baud rate for s1 boot
 0 - Exit menu and continue
->3
```

When option 3 is selected, the following submenu is displayed.

```
--- CHANGE IP ADDRESS ---
 Device List:
    001  Enabled    Ethernet  [ENET]
               local=7.1.1.5  remote=7.1.1.4  hwaddr=1000abcdef55
    004  Disabled   Serial Port 2 [S2]
               local=8.1.1.5  remote=8.1.1.4  hwaddr=ffffffffffff
 ---------------------------
select device to change ->
```

Select the appropriate device.

```
select device to change ->1             [Selects Ethernet]
```

When a valid device is selected, the following submenu is displayed.

```
 1 - Change local address
 2 - Change remote address
 0 - Return to main menu
->
```

Make the appropriate selection. To change the board’s IP address, you would select option 1, Change local address.

```
->1                                     [Selects the local address]
Current IP address = (7.1.1.5           [Displays the current value]
Enter new IP address ->Enter IP address in dot notation (e. g., 8.1.1.2)
```

Now enter the new IP address in dotted decimal notation.

7.1.1.5

After the selection has been entered, the new configuration is displayed, followed by the main menu.

```
    --- Device Configuration ---
    Power-On Test Devices:
       000  Disabled   System Memory [RAM]
       001  Enabled    Ethernet  [ENET]
       004  Enabled    Serial Port 2 [S2]
    ----------------------------
    Boot Sources:
       001  Enabled   Ethernet  [ENET]
                 local=7.1.1.5  remote=7.1.1.4  hwaddr=1000abcdef55
       004  Disabled  Serial Port 2 [S2]
                 local=8.1.1.5  remote=8.1.1.4  hwaddr=ffffffffffff
       005  Enabled   Serial Port 1 [S1]
                 Baud = 9600
    ----------------------------
    Debugger : Disabled
    ----------------------------
     1 - Enable/disable tests
     2 - Enable/disable boot devices
     3 - Change IP addresses
     4 - Ping test
     5 - Toggle ROM monitor debugger
     6 - Toggle automatic menu
     7 - Display configuration
     8 - Save changes to configuration
     9 - Set baud rate for s1 boot
     0 - Exit menu and continue
->
```

This option should be repeated to set all of the IP addresses to their appropriate values. If the suggested IP addresses are being used, the local and remote addresses for both the Ethernet and the Serial Port should match those in the above menu. Remember to save any configuration changes via option 8.

7.6.5 Using the Ping Test

Option four in the main menu selects the ping test. The ping test can be used for a basic assurance test of IP connectivity to the host workstation. It should be performed after setting the IP addresses to insure host-to-EVB communications. If the ping test fails, users can not load applications on to the board. The local and remote addresses for the specified device are used for the source and destination of the ICMP ping packets.

```
 1 - Enable/disable tests
 2 - Enable/disable boot devices
 3 - Change IP addresses
 4 - Ping test
 5 - Toggle ROM monitor debugger
 6 - Toggle automatic menu
 7 - Display configuration
 8 - Save changes to configuration
 9 - Set baud rate for s1 boot
 0 - Exit menu and continue
->4
```

When option 4 is selected, the current configuration is displayed, followed by another command prompt.

```
--- PING TEST ---
 Device List:
     001  Enabled   Ethernet  [ENET]
               local=7.1.1.5  remote=7.1.1.4  hwaddr=1000abcdef55
     004  Disabled  Serial Port 2 [S2]
                                local=8.1.1.5  remote=8.1.1.4
hwaddr=ffffffffffff
 ---------------------------
select device to ping ->
```

Select the appropriate device to ping (in this case only Ethernet is enabled).

```
select device to ping ->1               [selects the Ethernet port]
```

If the board is able to successfully ping the host, a message similar to the following should appear.

```
Using [ENET] to ping. press any key to stop.
PING 7.1.1.4 56 data bytes
78 bytes from 7.1.1.4: icmp_seq=0 ttl=255 time=2 ms
78 bytes from 7.1.1.4: icmp_seq=2 ttl=255 time=1 ms
```

Hitting any key terminates the ping test. The main menu is redisplayed following the PING status report.

```
--- 7.1.1.4 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
 1 - Enable/disable tests
 2 - Enable/disable boot devices
 3 - Change IP addresses
 4 - Ping test
 5 - Toggle ROM monitor debugger
 6 - Toggle automatic menu
 7 - Display configuration
 8 - Save changes to configuration
 9 - Set baud rate for s1 boot
 0 - Exit menu and continue
->
```

If the ping test fails,

- Verify that the local and remote IP addresses are set correctly. The local IP address should be that of the board and the remote IP address should be that of the host. These IP addresses were assigned during host configuration (see earlier chapter).
- Verify that the cables are connected properly.
- If a local 10Base2 Ethernet network is being used, that is one being used exclusively by the board and the host, insure that **both** ends of the Ethernet cable have BNC “T” type connectors with a terminator at one end.
- Verify TCP/IP is running on the host.

Note - The ROM Monitor will not respond to an inbound ping test from the host unless the ROM Monitor is in Debug mode (via options 5 and 0) or the ROM Monitor ping test is active on the EVB at the same time (via option 4).

7.6.6 Entering the Debugger

Option 5 toggles the feature of the ROM Monitor that allows communication with the host based source level debugger. Debugging may be enabled/disabled, and saved as part of the configuration using option 8. The debugger is not actually called by the monitor until after the user exits the main menu by selecting option 0 (exit and continue).

```
   --- Device Configuration ---
   Power-On Test Devices:
      000  Disabled   System Memory [RAM]
      001  Enabled    Ethernet  [ENET]
      004  Enabled    Serial Port 2 [S2]
   ----------------------------
   Boot Sources:
      001  Enabled   Ethernet  [ENET]
                local=7.1.1.5  remote=7.1.1.4  hwaddr=1000abcdef55
      004  Disabled  Serial Port 2 [S2]
                local=8.1.1.5  remote=8.1.1.4  hwaddr=ffffffffffff
      005  Enabled   Serial Port 1 [S1]
                Baud = 9600
   ----------------------------
   Debugger : Disabled
   ----------------------------
    1 - Enable/disable tests
    2 - Enable/disable boot devices
    3 - Change IP addresses
    4 - Ping test
    5 - Toggle ROM monitor debugger
    6 - Toggle automatic menu
    7 - Display configuration
    8 - Save changes to configuration
    9 - Set baud rate for s1 boot
    0 - Exit menu and continue
->5
ROM monitor debugger will be active on exit
   1 - Enable/disable tests
   2 - Enable/disable boot devices
   3 - Change IP addresses
   4 - Ping test
   5 - Toggle ROM monitor debugger
   6 - Toggle automatic menu
   7 - Display configuration
   8 - Save changes to configuration
   9 - Set baud rate for s1 boot
   0 - Exit menu and continue
->7

 --- Device Configuration ---
  Power-On Test Devices:
     000  Disabled   System Memory [RAM]
```

```
   001  Enabled    Ethernet  [ENET]
   004  Enabled    Serial Port 2 [S2]
  ----------------------------
  Boot Sources:
   001  Enabled   Ethernet  [ENET]
            local=7.1.1.5  remote=7.1.1.4  hwaddr=1000abcdef55
   004  Disabled  Serial Port 2 [S2]
            local=8.1.1.5  remote=8.1.1.4  hwaddr=ffffffffffff
   005  Enabled   Serial Port 1 [S1]
            Baud = 9600
  ----------------------------
  Debugger : Enabled (on exit)
  ----------------------------
   1 - Enable/disable tests
   2 - Enable/disable boot devices
   3 - Change IP addresses
   4 - Ping test
   5 - Toggle ROM monitor debugger
   6 - Toggle automatic menu
   7 - Display configuration
   8 - Save changes to configuration
   9 - Set baud rate for s1 boot
   0 - Exit menu and continue
->0
PowerPC ROM Monitor Debugger

                                                                  Waiting
for debug command...

Press any key to exit
```

Use option 8 to save the state of the ROM Monitor debugger. This option in combination with option 6, “Toggle automatic menu”, can be used to configure the EVB to automatically wait for the debugger to attach after power-on.

After enabling the ROM Monitor debugger (via option 5) and selecting option 0, the RISCWatch debugger can be started on the host and used to load an application onto the EVB. This is assuming the RISCWatch environment file has been updated for ROM Monitor communications. Once loaded successfully, the application can be run from the debugger.

The *RISCWatch Debugger User’s Guide* contains more information on how to use the debugger to load and execute files with the ROM Monitor as a non-JTAG target. At this point, it is recommended that users become familiar with the debugging environment by following the “Quick Start” sample debug session in the debugger’s User’s Guide. This session takes a user through the basics, including how to use the debugger to load and run applications on the board.

7.6.7 Disabling the Automatic Display

Option 6 in the main menu disables the automatic monitor display when the EVB boots up. After option 6 has been selected and the configuration has been saved (via Option 8), the menu display is disabled but continues to function until the user exits from the main menu. Following the next power-on or reset, the menu is no longer automatically displayed. This allows the user's image to be downloaded automatically with no menu input required. This feature also allows a user to download an application with no cable connected to the serial port 1 on the EVB (that is, without a terminal emulator).

After the automatic menu display has been disabled, the main menu can be accessed (assuming a terminal emulator is attached successfully to SP1 on the EVB) by pressing any key during the first five seconds that the EVB is booting. Otherwise, application download processing starts without displaying the main menu.

7.6.8 Displaying the Current Configuration

Option 7 displays the current configuration.

```
  1 - Enable/disable tests
  2 - Enable/disable boot devices
  3 - Change IP addresses
  4 - Ping test
  5 - Toggle ROM monitor debugger
  6 - Toggle automatic menu
  7 - Display configuration
  8 - Save changes to configuration
  9 - Set baud rate for s1 boot
  0 - Exit menu and continue
->7

 --- Device Configuration ---
  Power-On Test Devices:
     000  Disabled   System Memory [RAM]
     001  Enabled    Ethernet  [ENET]
     004  Enabled    Serial Port 2 [S2]
  ----------------------------
  Boot Sources:
     001  Enabled   Ethernet  [ENET]
               local=7.1.1.5  remote=7.1.1.4  hwaddr=1000abcdef55
     004  Disabled  Serial Port 2 [S2]
               local=8.1.1.5  remote=8.1.1.4  hwaddr=ffffffffffff
     005  Enabled   Serial Port 1 [S1]
               Baud = 9600
  ----------------------------
  Debugger : Enabled (on exit)
  ----------------------------
   1 - Enable/disable tests
   2 - Enable/disable boot devices
   3 - Change IP addresses
   4 - Ping test
   5 - Toggle ROM monitor debugger
   6 - Toggle automatic menu
   7 - Display configuration
   8 - Save changes to configuration
   9 - Set baud rate for s1 boot
   0 - Exit menu and continue
 ->
```

When a menu operation is selected to alter configuration settings, the current configuration is automatically redisplayed.

7.6.9 Saving the Current Configuration

Option 8 saves the current configuration for subsequent power-ons/resets..

```
  1 - Enable/disable tests
  2 - Enable/disable boot devices
  3 - Change IP addresses
  4 - Ping test
  5 - Toggle ROM monitor debugger
  6 - Toggle automatic menu
  7 - Display configuration
  8 - Save changes to configuration
  9 - Set baud rate for s1 boot
  0 - Exit menu and continue
->8
Configuration has been saved
  1 - Enable/disable tests
  2 - Enable/disable boot devices
  3 - Change IP addresses
  4 - Ping test
  5 - Toggle ROM monitor debugger
  6 - Toggle automatic menu
  7 - Display configuration
  8 - Save changes to configuration
  9 - Set baud rate for s1 boot
  0 - Exit menu and continue
  ->
```

The configuration is saved in the NVRAM on the evaluation board and is retained until a new configuration is subsequently saved.

7.6.10 Setting the Baud Rate for S1 Boots

Option 9 provides a mechanism for setting the baud rate to be used by serial port 1 when it is used as a device to download programs. Downloading over serial port 1 requires the use of a VT100 terminal emulator that supports **kermit** binary file transfer over serial port 1. RS/6000 and Sun users should note that the TIP terminal emulator does not support kermit binary file transfers. Windows 3.1 users can use the Windows Terminal program to perform kermit binary file transfers, but the baud rate is limited to 19 200. Windows 95 users can use HyperTerminal to perform kermit file tranfers at upto 115 200 baud. The kermit terminal emulator, available as shareware from the **http://www.columbia.edu/kermit** Internet site, can be used on any of the supported hosts to download programs over serial port 1 at speeds upto 115 200 baud.

```
--- Device Configuration ---
 Power-On Test Devices:
    000  Disabled   System Memory [RAM]
    001  Enabled    Ethernet  [ENET]
    004  Enabled    Serial Port 2 [S2]
 ----------------------------
 Boot Sources:
    001  Enabled   Ethernet  [ENET]
              local=7.1.1.5  remote=7.1.1.4  hwaddr=1000abcdef55
    004  Disabled  Serial Port 2 [S2]
              local=8.1.1.5  remote=8.1.1.4  hwaddr=ffffffffffff
    005  Enabled   Serial Port 1 [S1]
 ----------------------------
 Debugger : Enabled (on exit)
 ----------------------------
  1 - Enable/disable tests
  2 - Enable/disable boot devices
  3 - Change IP addresses
  4 - Ping test
  5 - Toggle ROM monitor debugger
  6 - Toggle automatic menu
  7 - Display configuration
  8 - Save changes to configuration
  9 - Set baud rate for s1 boot
  0 - Exit menu and continue
->9

Select a baud rate for S1 boot
  1 -         9600
  2 -       19200
  3 -       28800
  4 -       38400
  5 -       57600
  6 -     115200
=>4

--- Device Configuration ---
```

```
  Power-On Test Devices:
     000  Disabled   System Memory [RAM]
     001  Enabled    Ethernet  [ENET]
     004  Enabled    Serial Port 2 [S2]
  ----------------------------
  Boot Sources:
     001  Enabled   Ethernet  [ENET]
               local=7.1.1.5  remote=7.1.1.4  hwaddr=1000abcdef55
     004  Disabled  Serial Port 2 [S2]
               local=8.1.1.5  remote=8.1.1.4  hwaddr=ffffffffffff
     005  Enabled   Serial Port 1 [S1]
                                Baud = 38400         [download baud rate
appears here]
  ----------------------------
  Debugger : Disabled (on exit)
  ----------------------------
   1 - Enable/disable tests
   2 - Enable/disable boot devices
   3 - Change IP addresses
   4 - Ping test
   5 - Toggle ROM monitor debugger
   6 - Toggle automatic menu
   7 - Display configuration
   8 - Save changes to configuration
   9 - Set baud rate for s1 boot
   0 - Exit menu and continue
->
```

Use Option 8 to save the selected speed after reset and power-on.

7.6.11 S1 Boot

To perform an S1 boot you must have a terminal emulator which supports kermit file transfer. The file must be a valid boot image and must be sent in binary mode. If you have selected to use a baud rate other than 9600, you must set the terminal emulator to run at that speed before loading the file and set the speed back to 9600 after the down-load is complete. The following example shows loading the “usr_samp.img” file.

```
--- Device Configuration ---
Power-On Test Devices:
  000  Disabled  System Memory [RAM]
  001  Disabled  Ethernet [ENET]
  004  Disabled  Serial Port 2 [S2]
----------------------------
Boot Sources:
  001  Disabled  Ethernet [ENET]
                 local=7.1.1.5  remote=7.1.14  hwaddr=1000abcdef55
  004  Disabled  Serial Port 2 [S2]
                 local=8.1.1.5  remote=8.1.1.4  hwaddr=ffffffffffff
  005  Enabled   Serial Port 1 [S1]
                 Baud = 38400
----------------------------
Debugger: Disabled
----------------------------
 1 - Enable/disable tests
 2 - Enable/disable boot devices
 3 - Change IP addresses
 4 - Ping test
 5 - Toggle ROM monitor debugger
 6 - Toggle automatic menu
 7 - Display configuration
 8 - Save changes to configuration
 9 - Set baud rate for s1 boot
 0 - Exit menu and continue
->0
Booting from [S1] Serial Port 1...

PLEASE NOTE:  You must now...

       a. Exit from terminal emulation mode
       b. Modify the baud rate of your host session
       c. Transmit a file to the target in binary mode
       d. Reset the host baud rate to 9600
       e. Reenter terminal emulation mode
       f. Hit enter to execute the downloaded program
```

At this point kermit users must get to the terminal emulator command mode and change the line speed to match what was selected by option 9 and tell the terminal emulator to send the file in binary format.

^\c (Cntrl-\c)

```
(Back at waterdeep)
C-Kermit>set speed 38400
/dev/tty0, 38400 bps
C-Kermit>set file type bin
```

You can now load the file.

```
C-Kermit>send usr_samp.img
SF
Type escape character (^\) followed by:
X to cancel file,  CR to resend current packet
Z to cancel group, A for status report
E to send Error packet, Ctrl-C to quit immediately:

Sending: usr_samp.img => USR_SAMP.IMG
Size: 164864, Type: binary
................................................................
......
................................................................
.......
.... [OK]
ZB
```

When loading is completed, you must change the baud rate back to 9600 bps before continuing.

```
C-Kermit>set speed 9600
/dev/tty0, 9600 bps
```

After setting the baud rate back to 9600 bps, re-connect to your terminal emulator and press enter to complete the down-load.

```
C-Kermit>con
Connecting to /dev/tty0, speed 9600.
The escape character is Ctrl-\ (ASCII 28, FS)
Type the escape character followed by C to get back,
or followed by ? to see other options

Loaded successfully ...
Entry point at 0x22f20 ...

Hello 401 user!

Your ROM Monitor version is : 2.1

Your 604 Evaluation Board has 33554432 bytes of DRAM installed.

Your Ethernet controller’s network address is :  1000abcdef55

usr_samp done!
```

Assuming the S1 boot baud rate has been set to 38400 and option 0 has been selected to exit the ROM Monitor menu and initiate a load, Windows 95 HyperTerminal users can initiate the kermit binary file transfer by performing the following steps.

- Select **Call** and then **Disconnect.**
- Select **File, Properties, Configure** and set the baud to match the baud rate set via ROM Monitor option 9. In this case, it is 38400.
- Select **OK** and **OK** again.
- Select **Call** and then **Connect**.
- Select **Transfer, Send File** and type the filename of the file to load. Set the **Protocol** to Kermit.
- Select **Send**.

Upon successful completion of the transfer, the baud rate must be changed back to 9600.

- Select **Call** and then **Disconnect**.
- Select **File, Properties, Configure** and set the baud to 9600.
- Select **OK** and **OK** again.
- Select **Call** and then **Connect**.
- Hit **Enter** to complete the down-load sequence.

7.6.12 Exiting the Main Menu

Option 0 exits from the main menu, leaving the monitor active. If the debugger is active prior to selecting option 0, the ROM Monitor waits for the user to start the debugger on the host. In all other cases, option 0 initiates an attempt by the ROM Monitor to load an application from the host to the EVB over the enabled boot device(s). When downloading over the Ethernet or SLIP (S2), the host bootp and tftp configuration must be completed for the ROM Monitor to load successfully. Once loaded successfully, the application is executed.

When serial port 1 is used, the ROM Monitor requires the user to follow additional instructions to complete the download. The example shown here describes the sequence required when programs are downloaded over serial port 1.

```
--- Device Configuration ---
 Power-On Test Devices:
    000  Disabled   System Memory [RAM]
    001  Disabled   Ethernet  [ENET]
    004  Disabled   Serial Port 2 [S2]
 ----------------------------
 Boot Sources:
    001  Disabled  Ethernet  [ENET]
              local=7.1.1.5  remote=7.1.1.4  hwaddr=1000abcdef55
    004  Disabled  Serial Port 2 [S2]
              local=8.1.1.5  remote=8.1.1.4  hwaddr=ffffffffffff
    005  Enabled   Serial Port 1 [S1]
                               Baud = 38400
 ----------------------------
 Debugger : Enabled (on exit)
```

```
  ----------------------------
   1 - Enable/disable tests
   2 - Enable/disable boot devices
   3 - Change IP addresses
   4 - Ping test
   5 - Toggle ROM monitor debugger
   6 - Toggle automatic menu
   7 - Display configuration
   8 - Save changes to configuration
   9 - Set baud rate for s1 boot
   0 - Exit menu and continue
 ->0
Booting from [S1] Serial Port 1...

PLEASE NOTE: You must now...

     a. Exit from terminal emulation mode
     b. Modify the baud rate of your host session
     c. Transmit a file to the target in binary mode
     d. Reset the host baud rate to 9600
     e. Re-enter terminal emulation mode
     f. Hit enter to execute the downloaded program
```

The ROM Monitor will now wait for you to follow the above steps. The idea is that you must temporarily modify the terminal emulation session baud rate to match the baud rate expected by the ROM Monitor for the serial port 1 download. The file must then be transferred to the EVB from the host. The baud rate is restored to 9600 so that terminal emulation support can function after the program has been downloaded, The ROM Monitor will wait for you to restore the baud rate (9600) and hit enter prior to executing the downloaded program. This prevents any program I/O from being lost or incorrectly displayed when it begins execution.

The following is an example of what you might see when the program is allowed to run.

```
Loaded successfully ...
Entry point at 0x23130 ...
.
.
.
```

7.7 ROM Monitor User Functions

The ROM Monitor contains several functions that are available to user programs. The prototypes of these functions can be found in the usr_func.h file in the **/usr/osopen/PLATFORM/include** directory (**\osopen\PLATFORM\include** for PC users). These functions include:

- send_packet_on_bootdev() - allows an IP packet to be sent over the device that was used to load the application program (either the Ethernet or the second serial port, SP2).
- sh_register() - used to register a function that will be called when an IP packet is received by the ROM Monitor over the boot device.
- get_board_cfg() - reads the configuration data associated with the board.
- enet_send_macframe() - allows a frame to be sent over the Ethernet.
- enet_register() - allows the user to register an IP address for the Ethernet (an IP address different from that assigned to the ROM Monitor) and to specify a function to be called when a frame arrives for that address.
- enetisThere() - determines if the Ethernet chip is present on the board.
- enetInit() - initializes the Ethernet.
- getchar() - reads one character at a time from the keyboard buffer over the first serial port (SP1).
- s1putchar() - writes one character to the first serial port (SP1).

Applications must follow a predefined protocol to access ROM Monitor user functions. An example showing the proper calling procedures are included in the usr_samp.c sample program in the **samples** directory. This sample program calls the get_board_cfg() ROM Monitor function to determine the amount of DRAM installed on the board. This program will be run as a sample program in the next chapter.

7.8 Flash Update Utility

The **openbios/flash** directory contains all the code you need to re-program the flash memory on the EVB. This utility takes a binary image file targeted for the ROM as input, and generates a loadable file that will re-program the flash memory with the data in the binary input file. The file can then be loaded by an existing ROM Monitor version (which will be over-written upon successful completion of the loaded program) or via RISCWatch JTAG.

IMPORTANT: Please see the readme.txt file in the openbios/flash directory for important information regarding the use of this tool.

Be aware that if you use the ROM Monitor bootp or the RISCWatch ROM Monitor mode download process to re-program the flash, and the program loaded contains errors that will not allow you to download images in the same manner, your flash may be corrupted and rendered useless. In this case you will need to use RISCWatch JTAG or a ROM burner to re-program the flash.

RISCWatch JTAG users will find a RISCWatch command file, **rw_flash.cmd** in the openbios/flash directory. This command file can be used to prepare the EVB, load the flash update program containing the new binary image to program into the ROM, and start it running. This method can be used to program new flash parts, or to re-program a corrupted flash part when normal ROM Monitor downloads are not possible or inconvenient. When using this command file, RISCWatch **must** be used in JTAG mode.

8

401 EVB Sample Applications

This chapter describes the steps necessary to build and run the sample programs included in the 401 EVB software support package. This code is separate from ROM monitor code described in Chapter 7.

8.1 Overview

In the High C version of the EVB kit, the sample application programs are compiled, assembled, and linked using the IBM High C/C++ compiler, assembler, and linker. OS Open libraries are used during the link step to create an executable file in ELF format. This file includes the OS Open bootstrap code as well as other OS Open functions and is referred to as a boot file. One of the tools provided in the software support package, **eimgbld**, is then used to convert the boot file into the format used by the ROM Monitor to load programs onto the evaluation board (see Appendix B for more information on the ROM Monitor load format). The output of the **eimgbld** step is a file referred to as a boot image file.

There are several ways to load and execute a boot image file. One way is to use the ROM Monitor to load and execute the file. Network loads over Ethernet or SLIP require that the host contain the bootp and tftp servers and be properly configured to support the bootp and tftp protocols (see the previous chapters on host configuration and ROM Monitor setup). Loads over serial port 1 require a terminal emulator that supports the kermit transfer protocol. A ROM Monitor load is initiated via option 0 from the ROM Monitor main menu.

Another way to load and execute the boot image file is to use the RISCWatch debugger in ROM monitor mode. To bring up RISCWatch in ROM Monitor mode (see the *RISCWatch Debugger User's Guide* for details), you must update the RISCWatch environment file for ROM Monitor communications, enable the ROM Monitor debugger (via option 5), exit the ROM Monitor menu (via option 0) and then start up RISCWatch on the host system. The RISCWatch **load image** command can then be used to load the boot image file onto the board. Once loaded successfully, the **attach 42** and **logoff** commands can be issued to execute the program. The **attach 42** command informs the ROM Monitor that a process will be running and the **logoff** command tells the ROM Monitor to exit debug mode and start the execution of the program. After program execution, users should quit and restart RISCWatch before loading another boot image file to run. Without quitting RISCWatch, subsequent boot image execution can not be guaranteed. (Note: RISCWatch also provides the means to load a boot file (as opposed to a boot image file) via its **load file** command. See the "Running Your Programs" section in the *RISCWatch Debugger User's Guide* for additional information. This section also describes the steps required to load and debug boot and boot image files.)

8.2 ROM Monitor Flash Image

The flash memory on the EVB comes preprogrammed with a specific version of the ROM Monitor. This version may not be latest version of the ROM Monitor. To run the samples in the software support package, the latest version should be used. The latest version of the ROM Monitor is included in the software support package in the file:

- **/usr/osopen/PLATFORM/openbios/lib/rom_***.img** (RS6K & SUN)
- **\osopen\PLATFORM\openbios\lib\rom_***.img** (PC)

where *** is equal to the ROM Monitor version. If the *** version number of the ROM Monitor in the software support package does not match the version number displayed by the monitor when it comes up on the board, you can load the more recent version of the monitor provided in the software support package to re-program the flash memory.

The **rom_***.img** file can be loaded using the ROM Monitor or the RISCWatch debugger. For it to load properly upon the selection of ROM Monitor option 0, it must be copied to **boot.img** if the suggested bootptab entry was used (see the previous chapter on bootp configuration).

To load using RISCWatch, enable the ROM Monitor debugger (via option 5), exit the ROM Monitor menu (via option 0), start RISCWatch on the host system (make sure the RISCWatch environment file is setup for ROM Monitor communications), then use the following RISCWatch commands to load and execute the **rom_***.img** image file:

- **load image /usr/osopen/PLATFORM/openbios/lib/rom_***.img** (RS6K & SUN)
- **load image \osopen\PLATFORM\openbios\lib\rom_***.img** (PC)
- **attach 42**
- **logoff**

You will see screen information similar to that shown below. Lines preceded by "$$" are annotation for this example and do not appear on the screen.

```
$$ Standard ROM Monitor load screen below
401GF 1.2 ROM Monitor (9/5/95)
$$ Version 1.2 already installed corresponds to rom_12.img

  ------------- System Info -------------
  Processor speed   =  50 MHz
  Bus speed             =  25 MHz
  Amount of DRAM  =  8 MB
  -----------------------------------------

 --- Device Configuration ---
 Power-On Test Devices:
   000  Disabled  System Memory [RAM]
   001  Enabled   Ethernet [ENET]
   004  Enabled   Serial Port 2 [S2]
 ------------------------------------
 Boot Sources:
```

```
   001  Enabled   Ethernet [ENET]
                         local=7.1.1.5  remote=7.1.1.4
hwaddr=1000abcdef55
   004  Disabled  Serial Port 2 [S2]
                      local=8.1.1.5  remote=8.1.1.4  hwaddr=fffffffffffff
   004  Disabled  Serial Port 1 [S1]
                       Baud = 38400
 ----------------------------------
 Debugger: Disabled
----------------------------------
  1 - Enable/disable tests
  2 - Enable/disable boot devices
  3 - Change IP addresses
  4 - Ping test
  5 - Toggle ROM monitor debugger
  6 - Toggle automatic menu
  7 - Display configuration
  8 - Save changes to configuration
  9 - Set baud rate for s1 boot
  0 - Exit menu and continue
->0
$$ Selection of 0 causes evaluation board to be loaded. Previous
$$ arrangements must have been made to place the new ROM Monitor
$$ image (for ex. /usr/osopen/PLATFORM/openbios/lib/rom_13.img) in the
$$ place where bootp expects to find it (for ex. boot.img)
Booting from [ENET] Ethernet...
Sending bootp request ...


Loading file "/usr/osopen/PLATFORM/samples/boot.img" ...
Sending tftp boot request ...
Transfer Complete ...
Loaded successfully ...
Entry point at 0x10320 ...

$$ following information is from the ROM Monitor update program
################ IBM 4XX Evaluation Kit FLASH Update #################
                     ROM Monitor Version 1.3


$$ The universally administered hardware address for the Ethernet
$$ controller is kept in the flash ROM and is displayed here.
$$ Do not change this value for normal ROM Monitor updates
Network Address =
1000abcdef55


Do you wish to change Network Address? (y or n) n

$$ Heed the following warning. The ROM Monitor image could be
```

```
$$ rendered unusable and the board useless until the flash ROM is
$$ replaced.
     WARNING: You are about to re-program your ROM Monitor FLASH
              image.  Do NOT turn off power or press reset
              until this procedure is completed.  Otherwise
              the card may be permanently damaged!!!

Do you wish to continue? (y or n)y

Verifying new FLASH Image...
131072 matches, 0 mismatches

Update complete!
All done!
```

8.3 Using the Software Samples

The sample application programs are in **/usr/osopen/PLATFORM/samples** (**\osopen\PLATFORM\samples** for PC users). It is recommended that users first build and run the Dhrystone usr_samp, and timesamp sample programs as detailed below, to become familiar with the working environment. These sample programs use **basic_os.c** to provide a minimal OS Open configuration.

Additional details regarding the sample programs and application development in general can be found in the “Developing OS Open Applications” chapter in the *IBM OS Open User’s Guide*. That chapter should be referenced for instructions on building and running the applprog, benchmk, mailsamp, and cat sample programs.

The sample makefile contains the directives needed to build all the sample programs. It is suggested that this makefile be used as the starting point for building subsequent user applications.

Before attempting to build the samples, ensure the **osopen/bin** directory and the directory that contains the compiler, are part of your execution path (these steps should be modified accordingly based on where the compiler and the software support package were actually installed).

For **RS/6000** and **SUN** hosts :

- issue the command:

```
export PATH=$PATH:/usr/osopen/bin:/usr/highcppc/bin
```

OR (to update your PATH permanently)

- Edit **~/.profile** using an editor such as **vi**.
- Add `PATH=$PATH:/usr/osopen/bin:/usr/highcppc/bin` as a line in your profile before the line “export PATH”.
- Run **. ~/.profile** to update your profile.

For **PC** hosts:

- Edit AUTOEXEC.BAT using an editor such as e (you should back this file up before editing).
- If the following statement is missing, add it to the end of the file.

```
SET PATH=C:\highcppc\bin;C:\osopen\bin;%PATH%;
```

- Run AUTOEXEC.BAT to update your path.

NOTE: The "make" utility supplied with your evaluation kit may not run under a Windows NT command prompt that is started by "cmd.exe". To avoid potential problems, start a DOS command prompt using the command "COMMAND.COM" and compile from there. Also, some Windows 95 users may receive a ‘Program Requires MS-DOS Mode’ pop-up message when compiling. To prevent this annoying message from occurring, select ‘Properties’ for the MS-DOS window you are compiling from, then select ‘Advanced’ and ensure that the ‘Suggest MS-DOS mode as necessary’ box is not checked.

8.3.1 Building and Running the Dhrystone Benchmark

The Dhrystone benchmark is a commonly available integer benchmark. Since the main loop of this benchmark fits into the caches of many processors, its validity as a predictor of system performance may be suspect. It is included here as an example of an application to be built, loaded onto the evaluation board, and executed.

To build the Dhrystone benchmark, enter the command “**make dhry**” from the command line while in the **samples** directory. The makefile will compile the Dhrystone source files, link the resulting object files with the support libraries, and produce the boot file, **dhry,** and the boot image file, **dhry.img**.

If the bootptab entry suggested in the chapter on “Host Configuration” was used, then **dhry.img** must be renamed or copied to **boot.img** in order to be selected by the ROM Monitor load process. Select option 0 from the ROM Monitor screen to load and run the image.

To load using RISCWatch, enable the ROM Monitor debugger (via option 5), exit the ROM Monitor menu (via option 0), start RISCWatch on the host system (make sure the RISCWatch environment file is setup for ROM Monitor communications), then use the RISCWatch **load image** command to load the **dhry.img** file. Once successfully loaded, issue the **attach 42** and **logoff** commands to return control to the ROM Monitor and initiate the run.

You should see the following messages (or ones like them) appear on the ROM monitor screen. Explanations preceded by ## do not appear on the screen but are added here as clarification.

```
Booting from [ENET] Ethernet...
Sending bootp request ...
## This requests the Host workstation to return the name of the boot image

Loading file "/usr/osopen/PLATFORM/samples/boot.img" ...
Sending tftp boot request ...
## Having obtained the file name, the ROM monitor uses tftp to retrieve
the file from the
## host workstation
Transfer Complete ...
Loaded successfully ...
Entry point at 0x10238 ...
## Having loaded an image, the ROM monitor is now transferring control to
the application
## subsequent messages are from the application

Dhrystone Benchmark, Version 2.1 (Language: C)
Program compiled without 'register' attribute
Please give the number of runs through the benchmark:
```

At this point, enter the number of desired iterations. The test is designed not to give results if the selected iterations completes in less two seconds, so pick a large number (≥ 200000). After the test completes, a check screen will be displayed, followed by the benchmark results. The results may vary based on the system environment.

8.3.2 Building and Running the usr_samp Program

The **usr_samp.c** program is included as a sample to be built and run on the EVB. It's a simple program that shows how to properly call the get_board_cfg() ROM Monitor user function to determine the ROM Monitor version, the amount of DRAM installed on the board and the Ethernet controller's MAC address. Developers interested in using any of the ROM Monitor user functions should use this program as a guide.

To build the usr_samp program, enter the command "**make usr_samp**" from the command line while in the **samples** directory. The makefile will compile the usr_samp.c file, link the resulting object file with the support libraries, and produce the boot file, **usr_samp**, and the boot image file, **usr_samp.img**.

If the suggested bootptab was used, then **usr_samp.img** must be renamed or copied to **boot.img** in order to be selected by the Rom Monitor load process.Select option 0 from the ROM Monitor screen to load and run the image.

To load using RISCWatch, enable the ROM Monitor debugger (via option 5), exit the ROM Monitor menu (via option 0), start RISCWatch on the host system (make sure the RISCWatch environment file is setup for ROM Monitor communications), then use the RISCWatch **load image** command to load the **usr_samp.img** file. Once successfully loaded, issue the **attach 42** and **logoff** commands to return control to the ROM Monitor and initiate the run.

You should see the following messages (or ones like them) appear on the ROM Monitor screen.

```
Booting from [ENET] Ethernet...
Sending bootp request ...

Loading file “/usr/osopen/PLATFORM/samples/boot.img” ...
Sending tftp boot request ...
Transfer Complete ...
Loaded successfully ...
Entry point at 0x10180 ...

Hello 401 user!

Your ROM Monitor version is : 7.5

Your 401 Evaluation Board has 8388608 bytes of DRAM installed.

Your Ethernet controller’s network address is :  1000abcdef55

usr_samp done!
```

The DRAM amount listed should match the amount installed on the board.

8.3.3 Building and Running the timesamp Program

The **timesamp.c** program is included as a sample to be built and run on the EVB. This program is an example of how to properly time a particular function or benchmark. The user must know and define the time base frequency (the number of times the time base register is updated per second) in the timesamp.c to ensure the timing calculations are accurate.

To build the timesamp program, enter the command “**make timesamp**” from the command line while in the **samples** directory. The makefile will compile the timesamp.c file, link the resulting object file with the support libraries, and produce the boot file, **timesamp**, and the boot image file, **timesamp.img**.

If the suggested bootptab was used, then **timesamp.img** must be renamed or copied to **boot.img** in order to be selected by the Rom Monitor load process. Select option 0 from the ROM Monitor screen to load and run the image.

To load using RISCWatch, enable the ROM Monitor debugger (via option 5), exit the ROM Monitor menu (via option 0), start RISCWatch on the host system (make sure the RISCWatch environment file is setup for ROM Monitor communications), then use the RISCWatch **load image** command to load the **timesamp.img** file. Once successfully loaded, issue the **attach 42** and **logoff** commands to return control to the ROM Monitor and initiate the run.

You should see the following messages (or ones like them) appear on the ROM Monitor screen.

```
Booting from [ENET] Ethernet...
Sending bootp request ...

Loading file "/usr/osopen/PLATFORM/samples/boot.img" ...
Sending tftp boot request ...
Transfer Complete ...
Loaded successfully ...
Entry point at 0x10180 ...

Please give the number of runs through the benchmark:
```

At this point, enter the desired number of runs through the function or benchmark being timed. In this sample, the function being timed should execute for approximately a second, so a number between 1 and 10 would suffice.

8.4 Resolving Execution Problems

Configuration errors in the network or bootp tables cause most of the problems with running the sample applications. This section contains information that will aid users in identifying common problems.

8.4.1 Using the Ping Test on the ROM Monitor to Verify Connectivity

If the ping test fails, verify that TCP/IP is running on the host system and that the IP addresses on the selected interface are correct. The local address refers to the IP address of the evaluation board, and the remote refers to the host workstation address. The host workstation address must match the one selected during configuration of the host network interface. Also consult your TCP/IP documentation to insure proper network configuration.

8.4.2 bootp and tftp Servers (Daemons) for ROM Monitor loads

Insure that the bootp and tftp servers are started on the host workstation. If possible, use the **tftp** command from another workstation to retrieve the load image. If this fails, make sure the image exists in the target directory and that it is readable by “others”. If the tftp transfer succeeds, check the bootptab entry in the **bootptab** file to insure that it specifies the correct interface and IP address of the evaluation board.

8.5 Using OS Open Functions

OS Open provides the following major classes of functions for the embedded programming environment:

- Thread management
  The unit of execution context for OS Open is the thread as defined by POSIX standards. Functions are provided to create threads with various scheduling and execution attributes. To manage the execution environment, serialization and synchronization primitives are part of OS Open. The system also provides functions to associate data with specific threads.

- Storage management
  OS Open supports variable block allocations in the form of a heap. Functions are provided to extend the heap, query heap usage, and allocate storage to meet alignment constraints. OS Open also provides an independent storage management mechanism to allocate fixed blocks of storage in constant time.

- Interrupt and fault support
  OS Open provides functions to attach user-written code to any of the processor exceptions and interrupts. Most of the functions of OS Open can be used in these interrupt handlers, except for those functions that suspend execution or are valid only in the context of an executing thread. When the underlying hardware platforms support it, OS Open platform-specific libraries provide additional functions to attach user-written code to external interrupts supported on the platforms.

- Clock and timer management
  OS Open functions provide time-of-day clock support and the ability to create, use, and destroy timers. These timers can be one-time or periodic.

- Device support
  OS Open functions support the installation of user-written device drivers to provide character special files, block special files, and logical file systems. Low-level POSIX I/O (read, write) as well as ANSI C stream (fget, fput) functions are provided for device and regular file access.

- ANSI C library support
  OS Open provides a comprehensive set of ANSI C functions, providing support for string manipulation, memory management, string-to-number conversion, input/output, nonlocal jumps, and variable arguments.

- Pseudo device driver support
  OS Open provides several functions, such as TTY and DOS file system functions, that are installed and managed like device drivers, but they do not manipulate actual hardware nor do they have platform or device dependencies.

  OS Open provides functions that create and manage TCP/IP sockets. Network interface functions for Token Ring, Ethernet, and Serial Line Interface Protocol (SLIP) are also provided. With the TCP/IP protocol stack and network interfaces, additional functions are provided that implement several popular networking utilities, such as ping, ifconfig, ftp, and telnet.

- Debug functions and kernel abstract data types
  OS Open provides functions that set, clear, and query breakpoints. OS Open features an internal circular trace buffer for operating system and user events. Also, functions are provided that dump kernel data objects in a readable form.

9

Application Libraries and Tools

This chapter describes some of the application libraries and tools available in the EVB software support package. See the OS Open *User's Guide* and *Programmer's Reference* for additional information.

9.1 OS Open Libraries

The OS Open operating system comprises a real-time executive and optional libraries of functions and macros.

The real-time executive provides a operating system core for embedded applications. Depending on an application's requirements, an embedded application may also incorporate one or more optional libraries.

This modular approach enables embedded system developers to scale an OS Open operating system to match their application requirements. Because unneeded features are not present, an OS Open configuration can provide savings in system hardware, initialization and reset time, and program size.

Table 9-1 summarizes the OS Open libraries, described in the OS Open User's Guide and in this user's guide. For detailed descriptions of the OS Open functions and macros, refer to the *OS Open Programmer's Reference*.

Table 9-1. OS Open Libraries

Library	File Name	Platforms
Alignment Exception Support Library	alignLib.a	Common
ANSI C Library	cLib.a	Common
ANSI C Math Library	mathLib.a	Common
ANSI C I/O Library	fsLib.a	Common
Block Buffer Library	bbuffLib.a	Common
Bios Ethernet Library	benetLib.a	401 EVB
Boot Library(DRAM)	bootlLib.a	401 EVB
Boot Library(ROM)	bootrLib.a	401 EVB

Table 9-1. OS Open Libraries

Library	File Name	Platforms
ROM Monitor Ethernet Interface Library	benetLib.a	401 EVB
C++ runtime support (High C++™ support) Library	cppLib.a, crti.o, crtn.o, mwdctor.o	ELF
Card Services/enabler software layer for PCMCIA support	csLib.a	Common
Clock Support Library and NV-RAM	clockLib.a	401 EVB
Debug Support Library	dbLib.a	Common
Device and File Support Library	devLib.a	Common
DOS File System Support Library	fatLib.a	Common
Dynamic Loader Library	ldrLib.a	Common
Ethernet Support Library	enetLib.a	401 EVB
File Transfer Protocol Support Library	ftpLib.a	Common
Floating Point Library	fpeLib.a	401 EVB ELF
Floating Point Emulation Library	fpCSLib.a	Common
Input/output Support Library	ioLib.a	401 EVB
Kernel Abstract Data Types Library	kadtLib.a	Common
LCD Library	lcdLib.a	401 EVB
Network Support Library	netLib.a	Common
NFS Support Library	nfsLib.a	Common
OpenShell	shell.o	Common
PCMCIA ATA/IDE Hard disk device driver	pataLib.a	Common
PowerPC Low Level Access Support Library	ppcLib.a	401 EVB
Queue Library	queLib.a	Common
RAM Disk Library	ramdLib.a	Common
Rate Monotonic Scheduling (RMS) Library	rmsLib.a	Common
Remote Source Level Debug Library	rsldLib.a	Common
Ring Buffer Library	rngLib.a	Common
RPC Support Library	rpcLib.a	Common
Runtime Library	runlib.a	Common

Table 9-1. OS Open Libraries

Library	File Name	Platforms
SCSI Support Library	scsiLib.a	Common
Serial Support Library	asyncLib.a	401 EVB
Socket Services for PCMCIA support	ssLib.a	Common
Symbol Support Library	symLib.a	Common
TCP/IP Protocol Support Library	tcpipLib.a	Common
Telnet Daemon Support Library	tnetdLib.a	Common
Telnet Client Support Library	telnet.o	Common
The Real-time Executive	rtx.o, rtxLib.a	Common
OS Open Minimal Kernel	rtxmin.o	Common
OS Open Kernel Extensions for the minimal kernel	rtxext.o	Common
Timer Tick Support	tickLib.a	401 EVB
Trivial File Transfer Protocol	tftp.o	Common
TTY Support Library	ttyLib.a	Common

The real-time executive, the only required component in an OS Open operating system, provides a full set of basic operating system services.

- Thread management
- Storage management
- Signals
- Clocks and timers
- Interrupt and fault handling
- Message queues
- Semaphores
- Trace buffer support
- Miscellaneous services

The C functions for the real-time executive functions are in two libraries, **rtx.o** and **rtxLib.a**. The **rtx.o** library contains the OS Open real-time executive. The **rtxLib.a** library contains interface routines to OS Open functions, and is linked with application programs to resolve calls to the real-time executive.

9.2 Using Libraries and Support Software

The object libraries specific to the 401 EVB are described below.

Table 9-2. OS Open Libraries for the 401 EVB

Library	File Name
Boot Library(RAM)	bootlLib.a
Boot Library for OS Open in ROM	bootrLib.a
Ethernet Device Driver Support Library	enetLib.a
Input/Output Support Library	ioLib.a
LCD Library	lcdLib.a
PowerPC Low Level Access Support Library	ppcLib.a
Real-time Clock Interface Support Library with NV-RAM	clockLib.a
ROM Monitor Ethernet Interface Library	benetLib.a
Serial Support Library	asyncLib.a
Software Timer Tick Support Library	tickLib.a

9.2.1 Serial Port Support Library

This library supports the serial ports on the 401 EVB. Use in conjunction with the function provided by **devLib.a** and **fsLib.a** to provide a high level I/O interface to application programs. The serial port support functions reside in the **asyncLib.a** library.

9.2.2 Boot Library(RAM)

This library contains the OS Open bootstrap program for the appropriate platform. The boot library performs initial processing to prepare the completed application program for execution on the platform. For the 401 EVB, this processing includes moving the loaded program such that real addresses correspond with addresses assumed by the language development tools. The boot library for the 401 EVB also dynamically determines available heap space and prepares the symbol table for use by OS Open symbol management routines. The boot library does not export any functions.

9.2.3 Input/Output Support Library

The input/output functions reside in the **ioLib.a** library. To initialize the I/O subsystem, you must call **ioLib_init()** (normal mode) or **dbg_ioLib_init()** (ROM Monitor debug/ethernet) before performing any I/O other function.

9.2.4 PowerPC Low-Level Processor Access Support Library

The low-level access support library contains C-callable versions of the special PowerPC instructions. A few of the sample programs use these functions to manipulate the PowerPC 401GF's special registers. These functions provide access to processor instructions not generated by compilers. For example, device drivers often have a requirement to control data caching, disable interrupts, synchronize I/O, and other processor and platform-specific operations. The low-level access support functions reside in the **ppcLib.a** library.

9.2.5 ROM Boot Library

This library contains the OS Open bootstrap program and can be used instead of the boot library. The ROM boot library should be used when OS Open is being burnt into a ROM. The boot library performs initial processing to prepare the completed application program for execution on the platform. The boot library for the 401 EVB also dynamically determines available heap space and prepares the symbol table for use by OS Open symbol management routines. The boot library does not export any functions.

9.2.6 ROM Monitor Ethernet IP Interface Library

This library contains routines allowing access to the ROM Monitor's Ethernet IP interface. These functions allow the Ethernet to be simply configured with a unique IP address for use with TCP/IP functions. The ROM Monitor Ethernet IP Interface functions reside in **benetLib.a** library.

9.2.7 Real-time Clock Interface Support Library

This library contains routines to read and set the 401 EVB battery-backed real-time clock. These functions are not to be confused with the real-time clock functions provided directly by OS Open when the system is running. The real-time clock interface support functions reside in the OS Open's **clockLib.a** library and are available to perform the following features:

Set the OS Open clock from the real-time clock.

Set the real-time clock from user-supplied data.

Calibrate the real-time clock chip.

Read and write NV-RAM in the clock chip.

9.2.8 Integrated Ethernet IP Interface

This library provides the support for packet level interface to the integrated Ethernet interface in the 401 EVB. The Ethernet port support functions reside in the **enetLib.a** library.

9.2.9 Software Timer Tick Support Library

The OS Open system requires a periodic call to **timertick_notify()** to maintain internal clocks and timer functions. The **tickLib.a** library contains an implementation of the **timertick_notify()** function for PowerPC architecture machines. Timer tick support functions reside in the **tickLib.a** library.

9.3 Device Drivers Supplied with the 401 EVB

Device drivers provided with the 401 EVB include.

- Asynchronous
- Ethernet
- ROM Monitor Ethernet
- Liquid Crystal Display (LCD)

Examples and references are provided where appropriate.

For more information about any of the OS Open functions mentioned in this chapter, refer to the *OS Open Programmer's Reference*.

9.3.1 Asynchronous Device Driver

The asynchronous device driver supports the two asynchronous communication ports found on the 401GF EVB. Following is a brief functional description of the device driver.

- Support from 50 baud.
- Full duplex modem line control discipline.
- Overrun error, parity error, and framing error detection.
- BREAK interrupt detection.
- Support for data length of 5, 6, 7, and 8 bits.
- Support for 1, 1.5 and 2 stop bits.
- Support for receive and transmit parity.
- Support for odd and even parity.
- Support for transmitting BREAK.
- Support for 16 byte FIFO in the universal asynchronous receiver transmitter (UART).
- Programmed I/O (PIO) interrupt-driven slave communication.
- Interrupt driven input/output.
- Polled output functions.

Since only full duplex modem line control discipline is supported, connection between the asynchronous port and another device must be made through a "NULL" modem. A NULL modem is a device that crosses transmitted data and received data pins to enable communication. The only time a NULL modem is not necessary is when connection is made to a real modem device.

9.3.1.1 Device Driver Installation

The asynchronous device driver is installed by calling **driver_install()**. Following is an example of asynchronous device driver installation.

```
#include <sys/asyncLib.h>
int devhandle;
event_t event=10;
rc=driver_install(&devhandle, async_init, event);
```

The parameter *event* specifies the external interrupt id and must be between

EXT_PRIO_MIN and EXT_PRIO_MAX.

async_init() is declared in the file **<sys/asyncLib.h>** as follows.

```
int async_init(driver_t *dsw, va_list vargs)
```

Upon successful installation, **driver_install()** returns 0; otherwise –1 is returned. For more information on **driver_install()**, refer to the *OS Open Programmer's Reference*.

9.3.1.2 Device Installation

After the asynchronous device driver is installed, named devices can be created using **device_install()**. Following is an example of device installation.

```
#define S1DB_PARMS 1843200, (unsigned char *) 0x7E000000, 1,
EXT_IRQ_COM1
rc=device_install("/dev/s0", CHRTYPE, devhandle, 1, 128,
128,S1DB_PARMS);
```

For device installation, *devhandle* is the value obtained from the **driver_install()**. Device type CHRTYPE is defined in **<sys/devDrivr.h>**.

Additional parameters passed in the **device_install()** call are as follows.

Parameter	**Meaning**
Fourth Parameter	Port number to be installed (1)
Fifth Parameter	Size of write buffer
Sixth Parameter	Size of read buffer
Seventh Parameter	Input clock for the divisor
Eight Parameter	UART base register address 0x7E000000 or 0x7E000080
Ninth Parameter	UART register address delta, always 1
Tenth Parameter	Interrupt IRQ_MIN < event <IRQ_MAX (0 < event <15)

These are positional parameters.

Note: Write and read buffer sizes indicate number of characters that can be buffered in the device driver.

Upon successful installation, **device_install()** returns 0; otherwise –1 is returned. When the device is installed, error reporting for the device is turned off and xon/xoff pacing is enabled. For more information on **device_install()**, refer to the *OS Open Programmer's Reference*.

9.3.1.3 Opening Asynchronous Communication Ports

After the device is installed, the **open()** system call can be used to open a particular device. Following is an example of the **open()** system call used against the asynchronous port.

```
fd1=open("/dev/s0", O_RDWR, asyncParityNone, asyncParityOdd,
         asyncStopBits1, asyncDataBits8, 9600);
```

Additional parameters passed in **open()** are as follows.

Parameter	Meaning
First Parameter	Check/generate parity flag. Valid values are: asyncParityNone and asyncParityGen_Check
Second Parameter	Parity type. Valid values are asyncParityEven and asyncParityOdd. Because parameters are positional, this parameter must be specified even if parity is not used.
Third Parameter	Number of stop bits. Valid values are asyncStopBits1, asyncStopBits15 and asyncStopBits2. One and a half stop bits are only valid for data length of 5.
Fourth Parameter	Data length. Valid values are asyncDataBits5, asyncDataBits6, asyncDataBits7, and asyncDataBits8.
Fifth Parameter	Baud rate. Valid values range from 50 baud.

These are positional parameters. All parameter constants can be found in **<sys/ioctl.h>**.

Note: The *oflag* parameter, O_RDWR in this example, which is passed in the open call, is ignored by the device driver. When successful, **open()** returns a file descriptor, otherwise –1 is returned. **open()** can be called multiple times against the same asynchronous port. Communication parameters passed during the last **open()** call are set in the asynchronous port. For more information on **open()**, refer to the *OS Open Programmer's Reference*.

9.3.1.4 Reading and Writing

After successfully installing and opening the asynchronous port, **read()** and **write()** calls can be issued against that port. Multiple threads can issue **read()** and **write()** calls to the same port at the same time. However, simultaneous **read()** calls issued to the same port may block or be processed in an unexpected order. For these instances, thread scheduling and synchronization must be handled by the application.

Following is an example of **read()** and **write()** calls.

```
rc=write(fd1,"\nOS Open Real-time Executive\n", 29);
rc=read(fd1, buffer, 10);
```

fd1 is the value obtained from the **open()** call.

Note: For more information on **read()** and **write()**, refer to the *OS Open Programmer's Reference*.

9.3.1.5 I/O Control

An **ioctl()** call issued against asynchronous device driver accepts the commands listed in Table 9-3. All parameter constants can be found in **<sys/ioctl.h>**

Table 9-3. ioctl() Commands for Asynchronous Device Drivers

Command	Parameters	Explanation
ASYNCBAUDSET	Value from 50	Sets baud rate
ASYNCBAUDGET	Pointer to integer	Returns baud rate
ASYNCTRIGSET	asyncFifoTrigger1, asyncFifoTrigger4, asyncFifoTrigger8, asyncFifoTrigger14	Sets FIFO trigger level for asynchronous port
ASYNCTRIGGET	Pointer to integer	Returns current trigger level
ASYNCBREAKSET	None	Starts sending BREAK on port
ASYNCBREAKCLR	None	Stops sending BREAK on port
ASYNCSTICKGET	Pointer to integer	Returns the way the parity bit is interpreted by the port
ASYNCSTICKZERO	None	Disables stick parity
ASYNCSTICKONE	None	Parity interpretation tracks even/odd parity
ASYNCRERRORGET	Pointer to integer	Returns and clears read error conditions. Values are defined in asyncLib.h
ASYNCWERRORGET	Pointer to integer	Returns and clears write error conditions. Values are defined in asyncLib.h
ASYNCERROREN	None	Enables error reporting
ASYNCERRORDIS	None	Disables error reporting. All pending errors are cleared
ASYNCERRORGET	Pointer to integer	Returns error reporting enabled flag
ASYNCDLENGET	Pointer to integer	Returns current data length
ASYNCDLENSET	asyncDataBits5, asyncDataBits6, asyncDataBits7, asyncDataBits8	Sets data length
ASYNCSTOPGET	Pointer to integer	Returns number of stop bits

Table 9-3. ioctl() Commands for Asynchronous Device Drivers

Command	Parameters	Explanation
ASYNCSTOPSET1	None	Sets number of stop bits to 1
ASYNCSTOPSET1_5	None	Sets number of stop bits to 1.5
ASYNCSTOPSET2	None	Sets number of stop bits to 2
ASYNCPARITYNONE	None	Disable parity
ASYNCPARITYGEN	None	Enable parity
ASYNCPARITYSGET	Pointer to integer	Return parity status (enabled/disabled)
ASYNCPARITYODD	None	Sets parity to odd
ASYNCPARITYEVEN	None	Sets parity to even
ASYNCPARITYGET	Pointer to integer	Returns parity type
ASYNCXONENABLE	None	Enables XON/XOFF flow control
ASYNCXONDISABLE	None	Disables XON/XOFF flow control
ASYNCXONGET	Pointer to integer	Returns XON/XOFF flow control status
ASYNCMODEMSTAT	Pointer to integer	Returns modem status
ASYNCFLUSHIN	None	Flushes input buffer
ASYNCFLUSHOUT	None	Flushes output buffer
ASYNCDRAIN	None	Blocks until all characters in output buffer have been transmitted
ASYNCIGNBREAK	None	Ignores break interrupts
ASYNCSIGBREAK	None	Sends SIGINT on reception of break condition
ASYNCERRBREAK	None	Returns error from read upon reception of break condition. 0x00 is placed in the receive buffer at the position where break occurred.

Following is an example of an **ioctl()** call issued against an asynchronous device.

```
rc=ioctl(fd1, ASYNCXONDISABLE);
if (rc !=0) printf("ioctl failure\n");
```

fd1 is the value obtained from the **open()** call.

9.3.1.6 Polled Asynchronous I/O

A function is provided for polled output to s1 and s2 serial ports.

```
int s1dbprintf(unsigned long uart_clock, unsigned char *base_reg,
int reg_delta, event_t event, const char *format, ...)
int s2dbprintf(unsigned long uart_clock, unsigned char *base_reg,
int reg_delta, event_t event, const char *format, ...)
```

The parameters passed to these functions are identical to **printf()** except for *uart_clock*, *base_reg*, *reg_delta,* and event. *uart_clock* specifies the clock speed, *base_reg* specifies the address of the base UART register, *reg_delta* specifies the address space between UART registers, and *event* specifies the external interrupt level. Because polled I/O transmits characters synchronously, these functions may be called from first level interrupt handlers (FLIHs) or a user-supplied panic function. Since the function waits until the characters are actually sent before returning, use of this with long strings can significantly affect the timing of calling programs.

9.3.2 Ethernet Device Driver

The Ethernet device driver is a character device driver supporting packet level read/writes to the integrated Ethernet controller. The driver features the ability to open multiple files. Each file receives packets for a specific standard Ethernet or 802.3 address.

Function highlights are:

- Up to 8 receive channels
- Size of receive buffer pool determined by user at driver install time.

9.3.2.1 Device Driver Installation

enet_native_attach() attaches the TCP/IP protocol to the Ethernet device. Once the TCP/IP stack is attached, Ethernet packets can be sent/received using the TCP/IP functions or by using the Ethernet functions provided, such as **enet_send_packet().**

The Ethernet device is attached to the TCP/IP protocol stack after **tcpip_init()** and **net_init()** have been performed. The following is an example of attaching the TCP/IP protocol stack to the Ethernet.

```
#include <enetLib.h>
#define ENETHOST “401_board”
#define ENET_CONFIG “ ent0 401_board netmask 255.255.240.0 up”
#define SRAM_SIZE  8192  /* 8K buffer */
int do_enet()
{
 int rc;
 rc = tcpip_init( ENETHOST, 1, 1000); /* Initialize the TCP/IP
library */
 if(rc != 0)
   return -1;
 rc = net_init();  /*initialize netLib */
 if(rc != 0)
   return -1;
   /* attatch the TCP/IP protocol stack */
 rc = enet_native_attach( PROCESSOR_CLOCK_SPEED, SRAM_SIZE);
 r if(rc != 0)
   return -1;
 rc = ifconfig(ENET_CONFIG); /* configure network interface */
 if(rc != 0)
   return -1;
 return 0;
}
```

9.3.3 ROM Monitor Ethernet Device Driver

The ROM Monitor Ethernet device driver provides network access to the applications running on the 401 EVB, while still allowing the ROM Monitor to access the RISCWatch debugger over the ethernet.

This device driver uses code resident in the ROM monitor to send and receive ethernet packets. A different IP address must be specified to distinguish the packets from ROM Monitor and OS Open. I/O initialization should be done by calling **dbg_ioLib_init()** rather than **ioLib_init()**.

9.3.3.1 ROM Monitor Ethernet Installation and Initialization

The ROM Monitor Ethernet device driver is installed by calling **biosenet_attach()**. Following is a prototype of this function.

```
#include <benetLib.h>
int biosenet_attach(unsigned long ipaddr, int init_flag);
```

Upon successful installation, **biosenet_attach()** returns 0; otherwise -1 is returned. The IP address for OS Open is specified in the *ipaddr* parameter. The *init_flag* specifies whether the Ethernet controller needs to be initialized. If *init_flag* is set to 0 then the Ethernet controller is not initialized. If *init_flag* is set to a non-0 value, initialization of the Ethernet controller is performed.

9.3.4 Liquid Crystal Display Device Driver

The Liquid Crystal Display (LCD) device driver controls the optional LCD device. The device driver supports write and I/O control functions only.

9.3.4.1 LCD Device Driver Installation

The LCD device driver is installed by calling **driver_install()**. Following is an example of device driver installation.

```
#include <lcdLib.h>
int devhandle;
rc=driver_install(&devhandle, lcd_init);
```

lcd_init() is declared in the file **<lcdLib.h>** as follows.

```
int lcd_init(driver_t *dsw, va_list vargs)
```

Upon successful installation, **driver_install()** returns 0; otherwise –1 is returned. For more information on **driver_install()**, refer to the *OS Open Programmer's Reference*.

9.3.4.2 LCD Device Installation

After the LCD device driver is installed, a named device can be created using **device_install()**. Following is an example of device installation.

```
rc =device_install(“/dev/lcd”,CHRTYPE,devhandle, 2, 0x07E00300, 8);
```

For device installation, *devhandle* is the value obtained from the **driver_install()**. Device type CHRTYPE is defined in **<sys/devDrivr.h>**.

Additional parameters passed in the **device_install()** call are as follows:

Parameter	**Meaning**
Fourth Parameter	Number of lines on LCD (1 or 2)
Fifth Parameter	LCD base register address, typically 0x07E00300
Sixth Parameter	LCD register spacing, typically 1

These are positional parameters.

Upon successful installation, **device_install()** returns 0; otherwise –1 is returned. For more information on **device_install()**, refer to the *OS Open Programmer’s Reference*.

9.3.4.3 Opening LCD Device

After the device is installed, the **open()** system call can be used to open the LCD device for writing. Following is an example of the **open()** system call.

```
fd1=open("/dev/lcd”, O_WRONLY);
```

9.3.4.4 Writing to LCD

After successfully installing and opening the LCD, **write()** calls can be issued against the file descriptor.

Following is an example of a **write()** call.

```
rc=write(fd1,"\nOS Open Real-time OS\n", 23);
```

fd1 is the value obtained from the **open()** call.

For more information on **write()**, refer to the *OS Open Programmer’s Reference*.

9.3.4.5 I/O Control

An **ioctl()** call issued against LCD device driver accepts the commands listed in Table 9-4.

All parameter constants can be found in **<lcdLib.h>**

Table 9-4. ioctl() Commands for the LCD Device Driver

Command	Parameters	Explanation
LCD_MODE_DEC_CURSOR	None	Cursor decrements on write
LCD_MODE_DEC_DISPLAY	None	Cursor decrements but does not move, display shifts right
LCD_MODE_INC_CURSOR	None	Cursor increments on write
LCD_MODE_INC_DISPLAY	None	Cursor increments but does not move, display shifts left
LCD_MODE_NORMAL_SCROLL	None	Scrolls like a terminal
LCD_OFF	None	Nothing is displayed
LCD_ON	None	Only characters are displayed
LCD_ON_BLINK	None	Characters displayed, cursor character blinks
LCD_ON_CURSOR	None	Characters displayed, cursor displayed
LCD_ON_CURSOR_BLINK	None	Characters displayed, cursor displayed, cursor character blinks
LCD_INIT	None	Initialize display
LCD_CLEAR	None	Blanks, homes and unshifts LCD
LCD_DEC_CURSOR	None	Decrements and shifts cursor left
LCD_INC_CURSOR	None	Increments and shifts cursor right
LCD_DISPLAY_LEFT	None	Shift display left
LCD_DISPLAY_RIGHT	None	Shift display right
LCD_SET_CURSOR	int line, int column	Set cursor to line and column values
LCD_GET_CURSOR	int *line, int *column	Read cursor line and column position
LCD_GET_MODE	int *mode	Read current mode
LCD_GET_STATUS	int *status	Read current display status

Table 9-4. ioctl() Commands for the LCD Device Driver

Command	Parameters	Explanation
LCD_GET_SHIFT	int *shift	Read current shift amount

Following is an example of an **ioctl()** call issued against LCD device.

```
rc=ioctl(fd1, LCD_CLEAR);
if (rc !=0) printf("ioctl failure\n");
```

fd1 is the value obtained from the **open()** call.

9.4 Environment Bringup and Initialization

The following section describes the processing that occurs when the evaluation board environment is initialized.

Upon power-up or reset the ROM Monitor initializes the processor and other peripherals on the board. If a ROM Monitor load is attempted (via option 0), all enabled power-on tests are executed and, following their completion, a bootp request is sent to the host. This request involves an exchange of UDP packets corresponding to the bootp protocol. In essence, the ROM Monitor asks for and is supplied with the name of the boot image file on the host workstation. **tftp** (Trivial File Transfer Protocol) is then initiated by the ROM Monitor to transfer the boot image to the evaluation board.

Once the file has been transferred, two simple checks are made. A "magic number" in the boot image's 32-byte header verifies that the image is one that can be loaded by the ROM Monitor (ie., a file created by the eimgbld tool - see appendix B for details of the load format). After the load is complete, control is transferred to the specified entry point in the boot image, which is in the bootstrap program.

When using RISCWatch's **load image** command to load a boot image file, the debugger strips off the file's 32-byte header and loads the remaining bytes of the file onto the board. The start address of the load is designated in bytes 4-8 of the header. Once loaded, the IAR register is set to the boot image's entry point as defined in bytes 16-19 of the header. This entry point is in the bootstrap code. See the "Running Your Programs" section in the *RISCWatch Debugger User's Guide* for additional information on loading files.

9.4.1 Board bootstrap

The source for OS Open's bootstrap code is included in the **samples\bootLib** directory. The bootstrap program performs the following functions:

1. Unpacks the boot image format, placing the **.text** and **.data** sections in the addresses specified at link time.
2. Modifies the kernel configuration block with new heap size and start address.
3. Sets the **.bss** section to zeros, in accordance with ANSI C requirements.

9.4.2 Environment Initialization

OS Open requires information about the system environment at initialization. The following source files, which are included with the samples, are used to supply that information and to establish the working environment.

- basic_os.c - contains pieces of config.c, io_init.c, panic.c, thread0.c, and utils.c to provide a minimal OS Open configuration.
- config.c - configures the OS Open kernel
- io_init.c - initializes OS Open's I/O subsystem
- network.c - configures the host names and addresses for your environment
- panic.c - provides a sample panic function
- thread0.c - configures various features of OS Open (networking, remote debugger, etc.)
- utils.c - provides some useful utilities such as dir() to produce a directory listing

Additional information can be found in the "Configuring the OS Open Operating System" and "Developing OS Open Applications" chapters in the *IBM OS Open User's Guide*.

9.5 Tools

Several host tools are provided to assist you in using the EVB support package or creating your own applications for the PowerPC 401GF. The tools can also be used for ROM program development.

9.5.1 elf2rom

elf2rom takes an ELF format executable file (from the linker/binder), extracts the text and data sections, and writes them to a binary file for use as input to a ROM programmer. This tool can be used by those who wish to modify the ROM Monitor source code and create a new flash memory binary file for use with a ROM programmer or the flash update utility included with EVB software.

Syntax:

elf2rom [-v] [-d] [-p] [-s size] [-i offset] [-o output_file] input_elf

Description:

The program takes the input file *input_elf*(which is assumed to be an ELF file output from the linker), extracts the text and data sections, and writes them to the file*, output_file*. There are several optional flags that can affect elf2rom processing. They are described below.

-v	The verbose flag causes information about the generated output file to be written to stderr at the completion of the utility. This information includes the sizes and origins of the various sections and entry point.
-d	The debug flag will cause the symbol information from the input ELF file to be included after the data section in the output binary file.
-p	The promotion flag causes the data section to be aligned on a full word boundary if possible. This alignment facilitates full word moves of data to the appropriate target address without causing alignment exceptions.
-s	The size flag causes the output binary file to be padded to a particular size. This option is useful if it is necessary to create binary files that are the same size as a target ROM device. Error messages are generated if the generated image exceeds the specified size.
-i offset	The info flag places an information block into the output binary file at the specified offset. Since this info block overlays what is currently in the file at the specified offset, space should be reserved for its placement. The info block contains the following fields.
long block_id	Magic Number 0xBFAB0030
long entry_point	entry point of image

long toc_ptr	used for XCOFF; not used for ELF
long text_size	size of text section in bytes also offset from beginning of image to data section
long text_p_addr	text origin address as generated in ELF module
long data_size	size of data section
long data_p_addr	data origin as specified in generated ELF module
long bss_size	size of bss section
long bss_p_addr	bss origin as specified in generated ELF module
long num_syms	number of symbols from symbol section only valid if debug flag is set)
long sym_p_addr	address of symbol table. Calculated as text origin + offset of symbols with created ROM image
long text_offset	offset of text section from beginning of original ELF file. This information is required by certain debuggers
-o output_file	Allows the specification of an output file name. The default name is input_elf.img.
input_elf	This is simply the ELF binary input file. (elf2rom only)

The following picture shows the relationship of the various sections in the produced output file. The figure assumes that the info block flag [-i] was specified with an offset of 0x00.

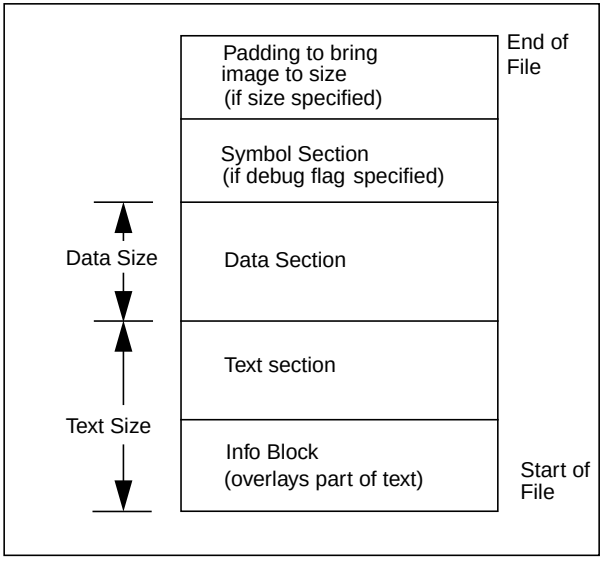


Figure 9-1. elf2rom Output File

Users can find an example of using elf2rom in the ROM Monitor's Makefile under **osopen/PLATFORM/openbios**.

9.5.2 hbranch

hbranch places a branch at the end of a ROM image. This simplifies production of ROM images for the PowerPC 401GF, which executes the instruction at the top location of memory following power-up or reset. **hbranch** can also be used to store a communication device's network address in the ROM's Vital Product Data (VPD) area.

Syntax:

hbranch [-v] [-s size] [-n net_addr] input_image

Description:

The program takes the input file *input_image* (which must be the output of elf2rom, or eimgbld with an information block at 0x0 relative) pads it to size *size* and writes a relative branch to the entry point recorded in the end of the image. The entry point must be a label, not a function descriptor. There are several optional flags that can affect **hbranch** processing. They are described below.

-v	The verbose flag causes information about the generated output image to be written to *stderr* at the completion of the utility. This information includes entry point information.
-s size	The size flag causes the image to be padded to a particular size. This facility is useful if it is necessary to create binary images that are the same size as a target ROM device.
-n net_addr	The network address flag stores net_addr, a 12 hex character network address (the media access control (MAC) address), in the VPD area in ROM. The ROM Monitor uses this option to store the EVB's ethernet controller's network address in its VPD.
-p patch_file	The patch file flag causes the file *patch_file* to be placed into the image just before the final branch and logically inserted into the instruction stream between the branch at the end of the file and the entry point. The patch file is inserted into the image "as is" and will usually contain the binary representation of position independent executable instructions. See Figure 9-2 for the details as to how normal hbranch processing is changed by a patch file.
input_image	This is simply the source image file. The output is written to *stdout.*.

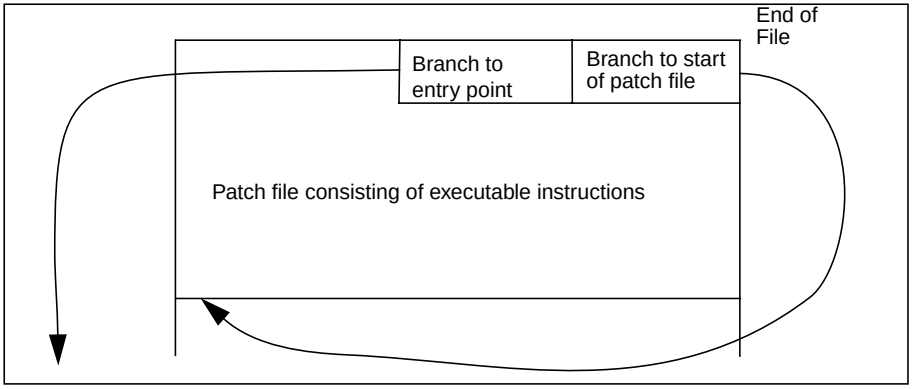


Figure 9-2. Detail of patch file placement

Figure 9-3 shows the relationship of the various sections in the produced output image.

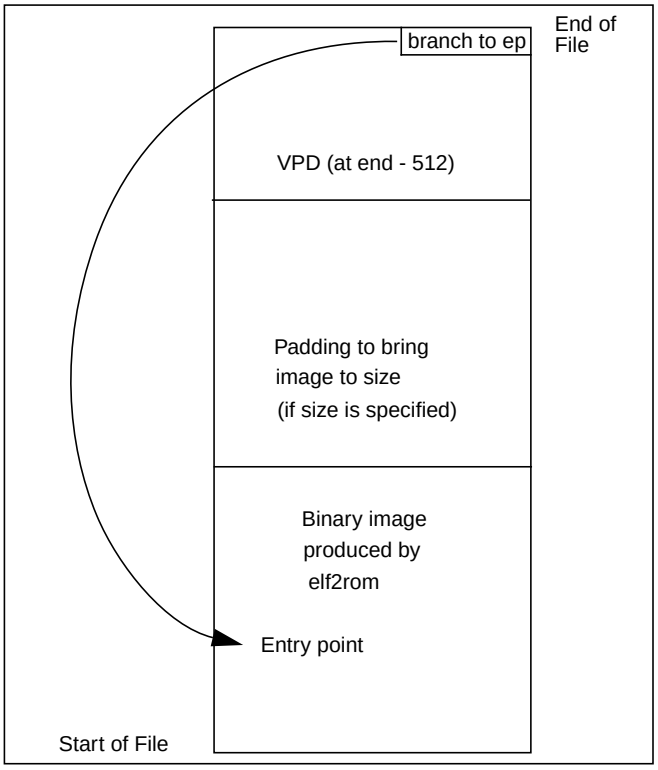


Figure 9-3. hbranch Output Image

Users can find an example of using hbranch in the ROM Monitor's Makefile under **osopen/PLATFORM/openbios**.

9.5.3 eimgbld

The **eimgbld** tool converts an output file from the linker/binder into the format used by the ROM Monitor to load programs from the host onto the evaluation board. The ELF file must be an otherwise executable file, with the text and data addresses bound at link time. Since the entry point of the ELF file will be used by the ROM loader, it must point to a suitable bootstrap.

Syntax:

eimgbld: [-D -P -S -v -b addr -m m_file -o o_file -s s_file -x x_file] input_elf

Description:

The program takes the input file *input_elf* (which must be the final ELF executable file produced from the build process) and converts it into the load format used by the ROM Monitor. There are several optional flags that can affect **eimbgld** processing. They are described below.

-D	Set debug flag. A flag is set in the image causing the ROM Monitor debugger to be invoked immediately after the image is loaded.
-P	Creates output image in PReP format. PReP format is used by some PowerPC platforms.
-S	Suppress symbol information. Specifying this flag will prevent the symbol table from being included in the image.
-v	Verbose option. Directs information about the produced image to stderr.
-b addr	Set the symbol start location to address, addr.
-m m_file	Specify the ROM address map file. The format of this file is two addresses on each line (start address and ending address separated by a “,”).
-o o_file	Allows the specification of an output file name. The default name is input_elf.img.
--s s_file	Restrict symbol table to names in specified file, s_name. The format of this file is one symbol on each line.
-x x_file	Suppress section names listed in specified file, x_name. The format of this file is one section name on each line.

Users can find an example of using eimgbld in the sample Makefile under **osopen/PLATFORM/samples**.

10

401 EVB Function Reference

This chapter describes the OS Open functions for the 401 EVB platform. The function calls and macros are arranged alphabetically by name. For information about the effective use of some of these functions, refer to the PPC401GF Embedded Controller User's Manual.

All descriptions contain the following sections:

- Synopsis
- Library
- Description
- Errors
- Attributes

Examples and references are provided or referenced where appropriate.

10.1 Attributes and Threads

Functions and macros have attributes that affect thread execution. Depending on their behavior, functions may or may not be "async safe," "cancel safe," and "interrupt handler safe."

10.1.1 Async Safe Functions

An async safe function may be entered by two or more concurrently executing threads, with each thread getting the correct results.

Functions that operate only on disjoint or local data objects are reentrant, and are therefore async safe. For example, **ppcCntlzw()** operates only on its arguments, making it reentrant and therefore async safe.

Functions that operate on common or global data objects may use serialization techniques, such as mutexes and semaphores, within the functions to ensure async safe operation. **enet_send_packet()** uses the functions **semwait()** and **sempost()** to force serialization. Refer to the *OS Open User's Guide* for more information about the use of mutexes and semaphores.

10.1.2 Cancel Safe Functions

The cancel safe attribute is important only to threads executing in deferred cancelability mode (the cancel state is enabled; the cancel type is deferred).

A thread executing in deferred cancelability mode can execute a cancel safe function without being canceled. If the same thread executes a non-cancel safe function, the thread may or may not be canceled during execution of the function.

10.1.3 Interrupt Handler Safe Functions

An interrupt handler safe function may be called by a first level interrupt handler (FLIH).

10.2 401 EVB Functions

Descriptions of the functions provided specifically to support the 401 EVB are listed in Table 10-1:

Table 10-1. Functions Specific to 401 EVB

Function or Macro	Description	Page
async_init()	Installs the asynchronous device driver	10-10
biosenet_attach()	Attaches the ROM Monitor Ethernet to an IP address	10-11
clock_set()	Sets the OS Open POSIX clock to the value obtained from the battery operated real time clock	10-13
clockchip_get()	Reads the real-time clock	10-14
clockchip_get_calibration( )	Returns the clock's calibration byte	10-15
clockchip_nvram_read()	Reads bytes from the clock chip's NVRAM	10-16
clockchip_nvram_write()	Writes bytes to the clock chip's NVRAM	10-17
clockchip_set_calibration()	Updates the clock's calibration byte with the supplied value	10-24

Table 10-1. Functions Specific to 401 EVB

Function or Macro	Description	Page
clockchip_start()	Starts the real-time clock	10-25
clockchip_stop()	Stops the real-time clock	10-26
clockLib_init()	Initializes the clockLib library routines	10-27
dbg_ioLib_init()	Initializes the I/O library	10-28
dcache_flush()	Flushes cache lines, beginning at the effective address and continuing for a specified number of bytes	10-29
dcache_invalidate()	Invalidates cache lines beginning at the effective address and continuing for a specified number of bytes	10-30
enet_disable_ipinput	Disables the forwarding of Ethernet packets to the TCP/IP protocol stack.	10-31
enet_enable_ipinput	Enables the forwarding of Ethernet packets to the TCP/IP protocol stack.	10-32
enet_native_attach	Attaches TCP/IP protocol stack.	10-33
enet_recv_packet()	Returns a pointer to the mbuf chain holding the packet received by the Ethernet device driver.	10-35
enet_send_packet()	Transmits packet over the Ethernet.	10-36
ext_int_disable()	Disables the interrupt level specified by an event	10-37
ext_int_enable()	Enables the interrupt level specified by an event	10-38
ext_int_install()	Installs a first level interrupt handler (FLIH) for an event.	10-39
ext_int_query()	Returns information about the FLIH	10-40

Table 10-1. Functions Specific to 401 EVB

Function or Macro	Description	Page
fpemul_init()	Installs floating point interrupt handler.	10-41
ioLib_init()	Initializes I/O library	10-42
lcd_init()	Installs the LCD device driver	10-43
ppcAbend()	Executes an invalid opcode forcing a program check interrupt	10-44
ppcAndMsr()	ANDs a value with the contents of the MSR	10-45
ppcCntlzw()	Counts consecutive leading zeros in a value	*10-46*
ppcDcbf()	Copies the cache block back to main storage (if the block resides in cache and has been modified with respect to main storage) and then invalidates the cache block	*10-47*
ppcDcbi()	Invalidates a cache block, discarding any modified contents if the block is valid in cache	10-48
ppcDcbst()	Copies a cache block, discarding any modified contents if the block is valid in cache	10-49
ppcDcbz()	Sets a cache block to 0	10-50
ppcDflush()	Writes 0's into the data cache and then turns data cache off by writing 0's into the data cache cacheability register (DCCR)	10-51
ppcEieio()	Ensures that all storage references before the call finish before any storage references after the call start	10-52
ppcHalt()	Is a one instruction spin loop, effectively putting the processor in an enabled wait at the point of invocation	10-53

Table 10-1. Functions Specific to 401 EVB

Function or Macro	Description	Page
ppcIcbi()	Invalidates an instruction cache block	10-54
ppcIsync()	Causes the processor to discard any instructions that may have been prefetched	10-55
ppdMfbear()	Returns the current value of the bus error address register (BEAR)	10-56
ppcMfbesr0()	Returns the current value of the bus error status register (BESR)	10-57
ppcMfbrcr0() - ppcMfbrcr7()	Return the value of their respective bus region control registers (BRCR0 - BRCR7)	10-58
ppcMfcdbcr()	Returns the value of the Cache Debug Control Register (CDBCR)	10-59
ppcMfdbcr()	Returns the value of the processor debug control register (DBCR)	10-60
ppcMfdbsr()	Returns the value of the processor debug status register (DBSR)	10-61
ppcMfdccr()	Returns the value of the Data Cache Cacheability Register (DCCR)	10-62
ppcMfdcwr()	Returns the value of the Data Cache Write-thru Register (DCWR)	10-63
ppcMfdear()	Returns the value of the Data Exception Address Register (DEAR)	10-64
ppcMfesr()	Returns the value of the exception syndrome register (ESR)	10-65
ppcMfevpr()	Returns the value of the exception vector prefix register (EVPR)	10-66
ppcMfgpr1()	Returns the current value of GPR(1)	10-67

Table 10-1. Functions Specific to 401 EVB

Function or Macro	Description	Page
ppcMfgpr2()	Returns the current value of GPR(2)	10-68
ppcMfiac1()	Returns the value of the instruction address compare register 1 (IAC1)	10-69
ppcMficcr()	Returns the value of the instruction cache cacheability register (ICCR)	10-70
ppcMficdbdr()	Returns the value of the Instruction Cache Debug Data Register (ICDBDR)	10-71
ppcMfiocr()	Returns the current value of the Input/Output configuration Register (IOCR)	10-72
ppcMfmsr()	Returns the value of the MSR	10-73
ppcMfpit()	Returns the value of the Programmable Interval Timer (PIT)	10-74
ppcMfpmcr0()	Returns the value of the Power management control register (PMCR0)	10-75
ppcMfpvr()	Returns the value of the processor version register	10-76
ppcMfsgr()	Returns the value of the Storage Guard Register (SGR)	10-77
ppcMfsler()	Returns the value of the Storage Little Endian Register (SLER)	10-78
ppcMfsprg0()- ppcMfsprg3()	Returns the value of the special purpose register generals (SPRG0 - SPRG3)	10-79
ppcMfsrr0()	Returns the value of SRR0	10-80
ppcMfsrr1()	Returns the current value of SRR1	10-81
ppcMfsrr2()	Returns the current value of SRR2	10-82

Table 10-1. Functions Specific to 401 EVB

Function or Macro	Description	Page
ppcMfsrr3()	Returns the current value of SRR3	10-83
ppcMftb()	Returns the current time base data	10-84
ppcMftsr()	Returns the current value of the timer status register (TSR)	10-85
ppcMtbrcr0() - ppcMtbrcr7()	Set the specified bus region control register (BRCR0 - BRCR7)	10-86
ppcMtcdbcr()	Sets the value of the Cache Debug Control Register (CDBCR)	10-87
ppcMtdac1() - ppcMtdac2()	Sets the values of the processor debug address compare registers (DAC1 - DAC2)	10-88
ppcMtdbcr()	Sets the value of the debug control register (DBCR)	10-89
ppcMtdbsr()	Sets the value of the debug status register (DBSR)	10-90
ppcMtdccr()	Sets the value of the Data Cache Cacheability Register (DCCR)	10-91
ppcMtdcwr()	Sets the value of the Data Cache Write-thru Register (DCWR)	10-92
ppcMtesr()	Sets the value of the exception syndrome register (ESR)	10-93
ppcMtevpr()	Sets the value of the exception vector prefix register (EVPR)	10-94
ppcMtiac1()	Sets the value of the instruction address compare register 1 (IAC1)	10-95
ppcMticcr()	Sets the value of the instruction cache cacheability register (ICCR)	10-96

Table 10-1. Functions Specific to 401 EVB

Function or Macro	Description	Page
ppcMtiocr()	Sets the input/output configuration register (IOCR)	10-97
ppcMtmsr()	Sets the MSR	10-98
ppcMtpit()	Sets the programmable interval timer (PIT)	10-99
ppcMtpmcr0()	Sets the value of the Power management control register (PMCR0)	10-100
ppcMtsgr()	Sets the value of the Storage Guard Register (SGR)	10-101
ppcMtsler()	Sets the value of the Storage Little Endian Register (SLER)	10-102
ppcMtsprg0() - ppcMtsprg3()	Sets the special purpose register generals (SPRG0 - SPRG3)	10-103
ppcMtsrr0()	Sets the SRR0	10-104
ppcMtsrr1()	Sets the SRR1	10-105
ppcMtsrr2()	Sets the SRR2	10-106
ppcMtsrr3()	Sets the SRR3	10-107
ppcMttb()	Sets the value of the current time base	10-108
ppcMttcr()	Sets the timer control register	10-109
ppcMttsr()	Sets the timer status register	10-110
ppcOrMsr()	Performs the OR of a value and the current MSR, updating the MSR	10-111
ppcSync()	Causes the processor to wait until all data cache lines scheduled to be written to main storage have actually been written	10-112

Table 10-1. Functions Specific to 401 EVB

Function or Macro	Description	Page
processor_speed()	Returns the internal clock speed of the 403 processor.	10-113
s1dbprintf()	A version of **printf()** that may be used before I/O has been established	10-114
s2dbprintf()	A version of **printf()** that may be used before I/O has been established for serial port 2	10-115
timertick_install()	Installs and starts the timer tick handler	10-116
timertick_remove()	Removes the timer tick handler	10-117
vs1dbprintf()	A version of **printf()** that uses polled writes (no interrupts), and may be used before I/O has been established and accepts a va_list as a parameter instead of a variable number of parameters	10-118

async_init()

Synopsis

#include <sys/asyncLib.h>

int driver_install(int *devhandle,async_init,...);

Library

asyncLib.a

Description

asyncLib.a is the asynchronous device driver that supports the asynchronous communication port on the 401 EVB platform. **asyncLib.a** is installed by calling **driver_install()** with *devhandle* as the first parameter and **async_init** as the second parameter.

Errors

None.

Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	No

References

- driver_install() : *OS Open Programmer's Reference*
- "Device Drivers Supplied with the 401 EVB" on page 9-7

Synopsis

#include <benetLib.h>

int biosenet_attach(unsigned long ipaddr, int init_flag);

Library

benetLib.a

Description

biosenet_attach() attaches the TCP/IP protocol stack to the ROM Monitor Ethernet . The IP address should be different from the IP address defined to the 401 EVB ROM Monitor. *init_flag* determines if **biosenet_attach()** should initialize the Ethernet interface. The Ethernet device should be initialized only if OS Open was loaded through an interface other than Ethernet. A non-0 value will cause **biosenet_attach()** to initialize the Ethernet and a 0 value causes **biosenet_attach()** not to initialize the Ethernet interface. **biosenet_attach()** returns 0 if successful and -1 if it is unsuccessful.

Note: When using **biosenet_attach()** the I/O should be initialized by calling **dbg_ioLib_init()** rather than **ioLib_init()**.

Errors

None.

Example

Initialize TCP/IP and define an IP address to **biosenet_attach()**.

```
#include<sys/tcpipLib.h>

int rc;

rc=tcpip_init("myhostname", 1 , 100);

if (rc!=0) {

return(-1);}

if (net_init() ) return(-1);

return(biosenet_attatch(0x07010104,0)); /* specify the
IP addr. and the init flag*/
```

Attributes

Async Safe	No
Cancel Safe	No
Interrupt Handler Safe	No

References

- “Ethernet Device Driver” on page 9-13

Synopsis

#include <clockLib.h>

int clock_set(void);

Library

clockLib.a

Description

clock_set() sets the OS Open POSIX clock to the value obtained from battery operated real time clock..

Errors

[EIO]	Real-time clock not running.

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

clockchip_get()

Synopsis

#include <clockLib.h>

int clockchip_get(time_t *timeval);

Library

clockLib.a

Description

clockchip_get() reads the battery-backed real-time clock into the *timeval* structure supplied by the user. The clockLib library must be initialized by calling **clockLib_init()** prior to calling this function.

Errors

[EINVAL]	Library not initialized.

Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- clockchip_set(), p. 10-18
- clockLib_init(), p. 10-27

Synopsis

#include <clockLib.h>

int clockchip_get_calibration(unsigned char *value);

Library

clockLib.a

Description

clockchip_get_calibration() returns the clock's calibration byte in the variable pointed to by *value*. The calibration byte occupies the five lower order bits of the byte. The sixth bit is a sign bit, "1" indicates positive calibration, "0" indicates negative calibration. The clockLib library must be initialized by calling **clockLib_init()** prior to calling this function.

Errors

[EINVAL]	Library not initialized.

Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- clockchip_set_calibration(), p. 10-24
- clockLib_init(), p. 10-27

clockchip_nvram_read()

Synopsis

#include <clockLib.h>

int clockchip_nvram_read(int index, unsigned char *buffer, int length);

Library

clockLib.a

Description

clockchip_nvram_read() reads non-volatile RAM from the clock chip. *index* specifies the starting byte of NVRAM, *buffer* points to the location where the bytes will be copied to and *length* specifies the maximum number of bytes to read. **clockchip_nvram_read()** returns the actual number of bytes read. The clockLib library must be initialized by calling **clockLib_init()** prior to calling this function.

Note: *index* must be within the range specified during **clockLib_init()**

Errors

[EINVAL]	Library not initialized or *index* out of range.

Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- clockchip_nvram_write(), p. 10-17
- clockLib_init(), p. 10-27

clockchip_nvram_write()

Synopsis

#include <clockLib.h>

int clockchip_nvram_write(int index, unsigned char *buffer, int length);

Library

clockLib.a

Description

clockchip_nvram_write() writes non-volatile RAM in the clock chip. *index* specifies the starting byte of NVRAM, *buffer* points to the location where the bytes will be copied from and *length* specifies the maximum number of bytes to write. **clockchip_nvram_write()** returns the actual number of bytes written. The clockLib library must be initialized by calling **clockLib_init()** prior to calling this function.

Note: *index* must be within the range specified during **clockLib_init()**

Errors

[EINVAL]	Library not initialized or *index* out of range.

Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- clockchip_nvram_read(), p. 10-16
- clockLib_init(), p. 10-27

clockchip_set()

Synopsis

#include <clockLib.h>

int clockchip_set(time_t timeval);

Library

clockLib.a

Description

clockchip_set() sets the battery-backed real-time clock to *timeval*, which should contain the number of seconds since January 1st, 1970 UTC.

Errors

[EIO]	Real-time clock not running.
[EINVAL]	Library not initialized.

Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

clock_set(), p. 10-13

Synopsis

#include <clockLib.h>

int clockchip_get(time_t *timeval);

Library

clockLib.a

Description

clockchip_get() reads the battery-backed real-time clock into the *timeval* structure supplied by the user. The clockLib library must be initialized by calling **clockLib_init()** prior to calling this function.

Errors

[EINVAL]	Library not initialized.

Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- clockchip_set(), p. 10-18
- clockLib_init(), p. 10-27

clockchip_get_calibration()

Synopsis

#include <clockLib.h>

int clockchip_get_calibration(unsigned char *value);

Library

clockLib.a

Description

clockchip_get_calibration() returns the clock's calibration byte in the variable pointed to by *value*. The calibration byte occupies the five lower order bits of the byte. The sixth bit is a sign bit, "1" indicates positive calibration, "0" indicates negative calibration. The clockLib library must be initialized by calling **clockLib_init()** prior to calling this function.

Errors

[EINVAL]	Library not initialized.

Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- clockchip_set_calibration(), p. 10-24
- clockLib_init(), p. 10-27

clockchip_nvram_read()

Synopsis

#include <clockLib.h>

int clockchip_nvram_read(int index, unsigned char *buffer, int length);

Library

clockLib.a

Description

clockchip_nvram_read() reads non-volatile RAM from the clock chip. *index* specifies the starting byte of NVRAM, *buffer* points to the location where the bytes will be copied to and *length* specifies the maximum number of bytes to read. **clockchip_nvram_read()** returns the actual number of bytes read. The clockLib library must be initialized by calling **clockLib_init()** prior to calling this function.

Note: *index* must be within the range specified during **clockLib_init()**

Errors

[EINVAL]	Library not initialized or *index* out of range.

Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- clockchip_nvram_write(), p. 10-17
- clockLib_init(), p. 10-27

clockchip_nvram_write()

Synopsis

#include <clockLib.h>

int clockchip_nvram_write(int index, unsigned char *buffer, int length);

Library

clockLib.a

Description

clockchip_nvram_write() writes non-volatile RAM in the clock chip. *index* specifies the starting byte of NVRAM, *buffer* points to the location where the bytes will be copied from and *length* specifies the maximum number of bytes to write. **clockchip_nvram_write()** returns the actual number of bytes written. The clockLib library must be initialized by calling **clockLib_init()** prior to calling this function.

Note: *index* must be within the range specified during **clockLib_init()**

Errors

[EINVAL]	Library not initialized or *index* out of range.

Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- clockchip_nvram_read(), p. 10-16
- clockLib_init(), p. 10-27

Synopsis

Library

Description

Errors

Attributes

Async Safe	No
Cancel Safe	No
Interrupt Handler Safe	No

- clockLib_init(), p. 10-27

clockchip_set_calibration()

Synopsis

#include <clockLib.h>

int clockchip_set_calibration(unsigned char value);

Library

clockLib.a

Description

clockchip_set_calibration() updates the clock's calibration byte with *value*. The calibration byte occupies the five lower order bits of the byte. The sixth bit is a sign bit, "1" indicates positive calibration, "0" indicates negative calibration. Adding bits speeds the clock up and subtracting bits slows the clock down. The clockLib library must be initialized by calling **clockLib_init()** prior to calling this function.

Errors

[EINVAL]	Library not initialized or too many bits in *value* byte.

Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- clockchip_get_calibration(), p. 10-15
- clockLib_init(), p. 10-27

Synopsis

#include <clockLib.h>

int clockchip_start(void);

Library

clockLib.a

Description

clockchip_start() starts the real-time clock. The clockLib library must be initialized by calling **clockLib_init()** prior to calling this function.

Errors

[EINVAL]	Library not initialized.

Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- clockchip_stop(), p. 10-26
- clockLib_init(), p. 10-27

clockchip_stop()

Synopsis

#include <clockLib.h>

int clockchip_stop(void);

Library

clockLib.a

Description

clockchip_stop() stops the real-time clock. The clockLib library must be initialized by calling **clockLib_init()** prior to calling this function.

Errors

[EINVAL]	Library not initialized.

Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- clockchip_start(), p. 10-25
- clockLib_init(), p. 10-27

Synopsis

#include <clockLib.h>

int clockLib_init(unsigned char *regbase, int reg_delta, int first_index, int last_index);

Library

clockLib.a

Description

clockLib_init() initializes the clockLib library routines. *regbase* specifies the base address of the clock/nvram chip, *reg_delta* specifies the distance (in bytes) between each addressable byte in the chip. *first_index* and *last_index* indicate the range of bytes in the NVRAM that can be accessed by **clockchip_nvram_read()** and **clockchip_nvram_write()**. The range is specified using starting and ending index values (inclusive). **clockLib_init()** returns 0 if successful.

Note: clockLib_init() should be called once at system initialization.

Errors

[EINVAL]	Already initialized or index out of range.

Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- clock_set(), p. 10-13
- clockchip_get(), p. 10-14
- clockchip_get_calibration(), p. 10-15
- clockchip_nvram_read(), p. 10-16
- clockchip_nvram_write(), p. 10-17
- clockchip_set(), p. 10-18
- clockchip_set_calibration(), p. 10-24
- clockchip_start(), p. 10-25
- clockchip_stop(), p. 10-26

dbg_ioLib_init()

Synopsis

#include <ioLib.h>

int dbg_ioLib_init(void);

Library

ioLib.a

Description

dbg_ioLib_init() initializes the I/O library. Unlike **ioLib_init()**, this function allows external I/O interrupts to be screened by the ROM monitor, enabling debug to be performed from outside of the OS Open environment. Only external I/O through IRQ's other than those used by the ROM Monitor are available to OS Open.

If successful, **dbg_ioLib_init()** returns 0. Otherwise, **dbg_ioLib_init()** returns –1.

Errors

[ENOMEM]	Insufficient memory to allocate first level interrupt handler control areas.

Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

ioLib_init(), p. 10-42

Synopsis

#include <ioLib.h>

void dcache_flush(void *address, unsigned int count);

Library

ioLib.a

Description

dcache_flush() flushes cache lines, beginning at the effective address and continuing for *count* bytes.

A cache line flush forces the current contents of the cache line to main storage (if the line is valid and marked as modified) and then invalidates the line.

Note: Since cache flushes occur on cache line boundaries, the operation can occur outside of the bounds specified by the function call. For example, if *address* is X'216' and *count* is X'12', two cache lines, spanning addresses from X'200' to X'23F', would be flushed.

Errors

None.

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- dcache_invalidate(), p. 10-30

dcache_invalidate()

Synopsis

#include <ioLib.h >

void dcache_invalidate(void *address, unsigned int count);

Library

ioLib.a

Description

dcache_invalidate() invalidates cache lines beginning at the effective address given by *address* and continuing for *count* bytes.

Note: Since cache invalidation occurs on cache line boundaries, invalidation can occur outside of the bounds implied by this command. For example, if *address* is X '104' and *count* is 16, the cache line spanning the addresses from X '100' to X '120' would be invalidated.

Errors

None.

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- dcache_flush(), p. 10-29

Synopsis

#include <enetLib.h>

void enet_disable_ipinput(void);

Library

enetLib.a

Description

enet_disable_ipinput() disables the forwarding of packets to the TCP/IP protocol stack. When forwarding is disabled Ethernet packets received by the Ethernet device driver can be read by the application using **enet_recv_packet()**.

Errors

None.

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- enet_native_attach(), p. 10-33
- enet_enable_ipinput(), p. 10-32
- enet_send_packet(), p. 10-36
- enet_recv_packet(), p. 10-35

enet_enable_ipinput()

Synopsis

#include <enetLib.h>

void enet_enable_ipinput(void);

Library

enetLib.a

Description

enet_enable_ipinput() enables the forwarding of packets to the TCP/IP protocol stack. When forwarding is enabled all Ethernet packets received by the Ethernet device driver are forwarded to the TCP/IP stack.

Errors

None.

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- enet_native_attach(), p. 10-33
- enet_disable_ipinput(), p. 10-31
- enet_send_packet(), p. 10-36
- enet_recv_packet(), p. 10-35

Synopsis

#include <enetLib.h>

int enet_native_attach(unsigned long processor_speed, unsigned long sram_size);

Library

enetLib.a

Description

enet_native_attach() attaches the TCP/IP protocol stack to the Ethernet device. The *processor_speed* specifies the CPU speed. The *sram_size* specifies the Ethernet controller's memory size. On the 401 EVB the *sram_size* parameter should be set to 8192. **enet_native_attach()** returns 0 if successful and -1 if it is unsuccessful.

Errors

None.

Example

The following is an example of initializing the TCP/IP protocol stack and attaching the Ethernet device.

```
#include <enet.h>
#define ENETHOST "401_board"
#define ENET_CONFIG " ent0 401_board netmask 255.255.240.0
up"
int do_enet()
{

int rc;
rc = tcpip_init( ENETHOST, 1, 1000); /* Initialize the
TCP/IP library */
if(rc != 0)
return -1;

rc = net_init();  /*initialize netLib */
if(rc != 0)
return -1;

rc = enet_native_attach(25000000,8 * 1024); /* attach the
tcp/ip proto. stack */
if(rc != 0)
return -1;
```

```
rc = ifconfig(ENET_CONFIG); /* configure network interface
*/
if(rc != 0)
return -1;

return 0;
}
```

Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	No

References

- enet_disable_ipinput(), p. 10-31
- enet_enable_ipinput(), p. 10-32
- enet_send_packet(), p. 10-36
- enet_recv_packet(), p. 10-35

Synopsis

#include <enetLib.h>

struct mbuf *enet_recv_packet(struct timespec *timeout);

Library

enetLib.a

Description

enet_recv_packet() returns a pointer to the mbuf chain holding the packet received by the Ethernet device driver. **enet_recv_packet()** will block waiting for packet reception for the maximum of time specified by *timeout*, If successful **enet_recv_packet()** returns a pointer to the mbuf chain containing the Ethernet packet, otherwise NULL is returned.

Errors

None.

Attributes

Async Safe	Yes
Cancel Safe	No
Interrupt Handler Safe	No
Callable from Application Thread Group	No

References

- enet_native_attach(), p. 10-33
- enet_disable_ipinput(), p. 10-31
- enet_enable_ipinput(), p. 10-32
- enet_send_packet(), p. 10-36

enet_send_packet()

Synopsis

#include <enetLib.h>

int enet_send_packet(struct ether_header *eh, struct mbuf *m, int total);

Library

enetLib.a

Description

enet_send_packet() transmits the packet described by *eh* and *m*. The Ethernet packet header is specified in *eh*. The destination, source hardware address, and packet type must be set in the *eh* structure prior to calling **enet_send_packet()**. *m* points to the mbuf chain that contains the actual packet. *total* must be set to the number of bytes to be transmitted. The value of *total* is set to the size of the Ethernet header plus the size of the packet contained in mbuf. If successful **enet_send_packet()** returns 0 otherwise -1 is returned. **enet_send_packet** will timeout after 3 seconds.

Errors

None.

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- enet_native_attach(), p. 10-33
- enet_disable_ipinput(), p. 10-31
- enet_enable_ipinput(), p. 10-32
- enet_recv_packet(), p. 10-35

Synopsis

#include <ioLib.h>

void ext_int_disable(int event);

Library

ioLib.a

Description

ext_int_disable() disables the interrupt level specified by *event.* The **ext_int_disable()** function returns nothing.

Errors

None.

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- ext_int_enable(), p. 10-38
- ext_int_install(), p. 10-39
- ext_int_query(), p. 10-40
- ioLib_init(), p. 10-42

ext_int_enable()

Synopsis

#include <ioLib.h>

void ext_int_enable(int event);

Library

ioLib.a

Description

ext_int_enable() enables the interrupt level specified by *event.*

ext_int_enable() returns nothing.

Errors

None.

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- ext_int_install(), p. 10-39
- ext_int_query(), p. 10-40
- ioLib_init(), p. 10-42

Synopsis

#include <flih.h>

#include <ioLib.h>

int ext_int_install(int event, flih_t *new_flih, flih_t *old_flih);

Library

ioLib.a

Description

ext_int_install() installs a first level interrupt handler (FLIH) for *event*.

If *new_flih* is NULL, the current interrupt handler is removed for the specified event. If *new_flih* is non-NULL, it points to a **flih_t** structure containing the following fields:

flih_stack	Pointer to the first stack location; obtained by allocating memory and adding the size of the stack. flih_stack must be 16 byte aligned.
flih_function	Pointer to a function invoked when *event* occurs.
arg	A user-defined (void *) value passed to *flih_function*.

If *old_flih* is not NULL, the previous values of *flih_function*, *flih_stack*, and *arg* are stored in the structure pointed to by *old_flih*.

If successful, **ext_int_install()** returns 0. Otherwise, **ext_int_install()** returns –1.

Errors

[EINVAL]	*event* does not refer to a valid event.

Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- ext_int_enable(), p. 10-38
- ext_int_query(), p. 10-40
- ioLib_init(), p. 10-42

ext_int_query()

Synopsis

#include <ioLib.h>

#include <flih.h>

int ext_int_query(int event, flih_t *flih);

Library

ioLib.a

Description

ext_int_query() returns information about the first level interrupt handler (FLIH), if any, for *event*.

The *flih* argument points to a **flih_t** structure containing the following fields:

flih_stack	Pointer to the first stack location; obtained by allocating memory and adding the size of the stack.
flih_function	Pointer to a function invoked when *event* occurs.
arg	A user-defined (void *) value passed to *flih_function*. If no FLIH is installed for the specified level, each field in the **flih_t** structure is assigned NULL.

If successful, **ext_int_query()** returns 0. Otherwise, **ext_int_query()** returns –1.

Errors

[EINVAL]	*event* does not refer to a valid event.

Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- ext_int_enable(), p. 10-38
- ext_int_install(), p. 10-39
- ioLib_init(), p. 10-42

Synopsis

#include <fpeLib.h>

void fpemul_init(void);

Library

fpCSLib.a

Description

fpemul_init() installs the floating point interrupt handler. **fpemul_init()** is only needed when floating point emulation is required with the XCOFF version of OS Open for PowerPC processors without floating point hardware. **fpemul_init()** is not required for floating point when running with the ELF version of OS Open. **fpemul_init()** returns nothing.

Errors

None.

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

ioLib_init()

Synopsis

#include <ioLib.h>

int ioLib_init(void);

Library

ioLib.a

Description

ioLib_init() initializes the I/O library.

If successful, **ioLib_init()** returns 0. Otherwise, **ioLib_init()** returns –1.

ioLib_init() should not be used on a 401 EVB when using the ROM Monitor Ethernet interface or the ROM monitor debugger. **dbg_ioLib_init()** should be used instead.

Errors

[ENOMEM]	Insufficient memory to allocate first level interrupt handler control areas.

Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes

Synopsis

#include <lcdLib.h>

int driver_install(int devhandle, lcd_init, int number_lines, int register_base, int register_delta);

Library

lcdLib.a

Description

lcdLib.a is the LCD device driver. It is installed by calling **driver_install()** with five parameters. The first parameter is the device handle, *devhandle*. The second parameter is the device driver initialization function, **lcd_init**. The third parameter is the number of lines contained on the display, *number_lines*, typically 2. The fourth parameter is the I/O address of the base register, *register_base*, typically 0x7E000300. The fifth parameter is the register spacing, *register_delta*, typically 1. For more information about **lcd_init** and **driver_install()**, refer to “Device Drivers Supplied with the 401 EVB” on page 9-7.

Errors

None.

Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	No

ppcAbend()

Synopsis

#include <ppcLib.h>

void ppcAbend(void)

Library

ppcLib.a

Description

ppcAbend() executes an invalid opcode forcing a Program Check interrupt.

Errors

None.

Example

- Force an illegal instruction exception.

```
ppcAbend()
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

unsigned long ppcAndMsr(unsigned long value);

Library

ppcLib.a

Description

ppcAndMsr() ANDs *value* with the contents of the MSR.

The MSR is updated with the result of the AND operation.

ppcAndMsr() returns the previous contents of the MSR.

Refer to the **<ppcLib.h>** file for the defines of the MSR constants.

Errors

None.

Example

- Disable external interrupts.

```
unsigned long orig_msr = ppcAndMsr(~ppcMsrEE);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- ppcOrMsr(), p. 10-111
- ppcMtmsr(), p. 10-98
- *PPC401GF Embedded Controller User's Manual*

ppcCntlzw()

Synopsis

#include <ppcLib.h>

unsigned long ppcCntlzw(unsigned long value);

Library

ppcLib.a

Description

ppcCntlzw() counts consecutive leading zeros in *value.*

ppcCntlzw() returns the count, which ranges from 0 through 32, inclusive.

Errors

None.

Example

- Return count of leading zeros in variable k.

```
int k;
 unsigned long k = ppcCntlzw(0x0700AA55); /* k = 5 */
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

void ppcDcbf(void *addr);

Library

ppcLib.a

Description

ppcDcbf() copies the cache block at the effective address specified by *addr* back to main storage (if the block resides in cache and has been modified with respect to main storage) and then invalidates the cache block.

Effectively, this function acts like **ppcDcbst()** followed by **ppcDcbi()**.

Errors

None.

Example

- Flush the cache line at the effective address X'1000' to main storage and then invalidate the cache line. You might do this in preparation for a DMA slave transfer.

```
ppcDcbf((void *)0x1000);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *ppcDcbst(), p. 10-49*
- *ppcDcbi(), p. 10-48*
- *ppcDcbz(), p. 10-50*
- *PPC401GF Embedded Controller User's Manual*

ppcDcbi()

Synopsis

#include <ppcLib.h>

void ppcDcbi(void *addr);

Library

ppcLib.a

Description

ppcDcbi() invalidates the cache block containing *addr*, discarding any modified contents if the block is valid in cache.

Errors

None.

Example

- Invalidate the cache line beginning with 0x3000. This might be done before reading an area of storage updated by a DMA transfer.

```
ppcDcbi((void *)0x3000);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *ppcDcbst(), p. 10-49*
- *ppcDcbi(), p. 10-48*
- *ppcDcbz(), p. 10-50*
- *PPC401GF Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

void ppcDcbst(void *addr);

Library

ppcLib.a

Description

ppcDcbst() copies the cache block containing *addr* to main storage, if the block is valid in cache and has been modified with respect to main storage.

Errors

None.

Example

- Force the cache line beginning with 0x4000 to memory if the block is valid and out of sync with storage. This would be done to synchronize the cache and storage without invalidating the cache line.

```
ppcDcbst((void *)0x4000);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *ppcDcbf(), p. 10-47*
- *ppcDcbi(), p. 10-48*
- *ppcDcbz(), p. 10-50*
- *PPC401GF Embedded Controller User's Manual*

ppcDcbz()

Synopsis

#include <ppcLib.h>

void ppcDcbz(void *addr);

Library

ppcLib.a

Description

ppcDcbz() sets the cache block containing the byte referenced by *addr* to 0.

The line is established, if necessary, without fetching the line from main storage.

Note: If an invalid real address is specified, problems could occur when a subsequent attempt is made by the cache unit to store that line to main storage.

Errors

None.

Example

- Assume buffer is 16 cache lines long and cache aligned. To quickly set it to 0, set to first buffer address.

```
char *bpt = buffer;
for(j = 0; j < 16; j++)
  {
   ppcDcbz((void *)bpt);
   bpt += cache_line_size;
 }
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *ppcDcbf(), p. 10-47*
- *ppcDcbi(), p. 10-48*
- *ppcDcbst(), p. 10-49*
- *PPC401GF Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

void ppcDflush(void);

Library

ppcLib.a

Description

ppcDflush() will write 0's into the data cache and then turn data cache off by writing 0's into the Data Cache Cacheability Register (DCCR).

Errors

None.

Example

- Force data reads from memory instead of from the data cache.

```
ppcDflush();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

ppcEieio()

Synopsis

#include <ppcLib.h>

void ppcEieio(void);

Library

ppcLib.a

Description

ppcEieio() ensures that all storage references before the call finish before any storage references after the call start.

Errors

None.

Example

- Ensure storage references are done in order.

```
char *one_loc = (char *)0x202;
char *two_loc = (char *)0x204;

*one_loc = 0xAA; /* write a 0xAA to 0x202 */
ppcEieio(); /* insure the store completes before
setting two_loc */
*two_loc = 0x55;
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

void ppcHalt(void);

Library

ppcLib.a

Description

ppcHalt() is a one instruction spin loop, effectively putting the processor in an enabled wait at the point of invocation.

Errors

None.

Example

- Wait at the point of invocation.

```
ppcHalt();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

ppcIcbi()

Synopsis

#include <ppcLib.h>

void ppcIcbi(void *addr);

Library

ppcLib.a

Description

ppcIcbi() invalidates the Instruction Cache Block pointed to by the address passed. This may be done after updating an instruction.

Errors

None.

Example

- Write a trap into location 0x3000.

```
unsigned in * i_addr = (int *) 0x3000;
*i_addr = 0x7c800008; /* tw instruction */
ppcDbcst((void *) 0x3000);
ppcIcbi((void *) 0x3000);
ppcIsync();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

void ppcIsync(void);

Library

ppcLib.a

Description

ppcIsync() causes the processor to discard any instructions that may have been prefetched before **ppcIsync()**. This call must be used after modifying instruction storage.

Errors

None.

Example

- Place a trap into a given address.

```
*trap_address = 0x7F000008;
 ppcIsync();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

ppcMfbear()

Synopsis

#include <ppcLib.h>

unsigned long ppcMfbear(void)

Library

ppcLib.a

Description

ppcMfbear() returns the current value of the Bus Error Address Register.

Errors

None.

Example

- After a machine check, retrieve the BEAR.

```
bear = ppcMfbear();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *ppcMfbesr0(), p. 10-57*
- *PPC401GF Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

unsigned long ppcMfbesr0(void);

Library

ppcLib.a

Description

ppcMfbesr0() returns the current value of the Bus Error Status Register, which identifies the nature of a bus error detected by the processor.

The file **<ppcLib.h>** defines constants for use with the BESR0.

Errors

None.

Example

- Retrieve bus error syndrome information.

```
besr = ppcMfbesr0()
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- ppcMfbear(), p. 10-56
- *PPC401GF Embedded Controller User's Manual*

ppcMfbrcr0() • ppcMfbrcr7()

Synopsis

#include <ppcLib.h>

unsigned long ppcMfbrcr0(void);

unsigned long ppcMfbrcr1(void);

unsigned long ppcMfbrcr2(void);

unsigned long ppcMfbrcr3(void);

unsigned long ppcMfbrcr4(void);

unsigned long ppcMfbrcr5(void);

unsigned long ppcMfbrcr6(void);

unsigned long ppcMfbrcr7(void);

Library

ppcLib.a

Description

ppcMfbrcr0() - ppcMfbrcr7() return the value of their respective Bus Region Control Register(BRCR0 - BRCR7). **ppcMfbrcr0() - ppcMfbrcr7()** are implemented as macro's**.** The file **<ppcLib.h>** has several constants defined for use with the BRCR registers.

Errors

None.

Example

- Retrieve the value of BRCR4.

```
unsigned long current_brcr4=ppcMfbrcr4();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

unsigned long ppcMfcdbcr(void);

Library

ppcLib.a

Description

ppcMfcdbcr() returns the value of the Cache Debug Control Register (CDBCR).

<ppcLib.h> has constants defined for use with the CDBCR register.

Errors

None.

Example

- Retrieve the current value of the CDBCR.

```
unsigned long cdbcr_value=ppcMfcdbcr();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

ppcMfdbcr()

Synopsis

#include <ppcLib.h>

unsigned long ppcMfdbcr(void);

Library

ppcLib.a

Description

ppcMfdbcr() returns the value of the processor debug control register (DBCR). The DBCR is used to enable debug events, reset the processor, control timer operations during debug events, and set the debug mode of the processor.

WARNING: Enabling bits 0 and 1 can cause unexpected results. Enabling bits 2 and 3 will cause a processor reset to occur. The DBCR is designed to be used by development tools, not applications.

Refer to the **<ppcLib.h>** for defined constants for the DBCR.

Errors

None.

Example

- Retrieve the value of DBCR register. A debugger would require the value of the DBCR.

```
unsigned long current_DBCR=ppcMfdbcr();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

unsigned long ppcMfdbsr(void);

Library

ppcLib.a

Description

ppcMfdbsr() returns the value of the processor debug status register (DBSR). The DBSR contains the status of debug events, the JTAG serial buffers, and the most recent reset.

The file **<ppcLib.h>** defines constants that can be used when referring to the DBSR.

Errors

None.

Example

- Retrieve the value of DBSR register. A debugger would require the value of the DBSR.

```
unsigned long current_DBSR=ppcMfdbsr();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

ppcMfdccr()

Synopsis

#include <ppcLib.h>

void ppcMfdccr(unsigned long dccr_value);

Library

ppcLib.a

Description

ppcMfdccr() returns the value of Data Cache Cacheability Register (DCCR).

Errors

None.

Example

- Set the value of the DCCR.

```
#include<ppcLib.h>
unsigned long dccr_value=ppcMfdccr();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	No

References

- *PPC401GF Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

unsigned long ppcMfdcwr(void);

Library

ppcLib.a

Description

ppcMfdcwr() returns the value of the Data Cache Write-thru Register (DCWR).

Errors

None.

Example

- Retrieve the current value of the DCWR.

```
unsigned long dcwr_value=ppcMfdcwr();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

ppcMfdear()

Synopsis

#include <ppcLib.h>

void ppcMfdear(unsigned long dear_value);

Library

ppcLib.a

Description

ppcMfdear() returns the value of Data Exception Address Register (DEAR).

Errors

None.

Example

- Set the value of the DEAR.

```
#include<ppcLib.h>
unsigned long dear_value=ppcMfdear();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

unsigned long ppcMfesr(void);

Library

ppcLib.a

Description

ppcMfesr() returns the value of the Exception Syndrome Register (ESR). Bits 7 to 31 are reserved.

Errors

None.

Example

- Get the ESR value.

```
esr_value= ppcMfesr();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

ppcMfevpr()

Synopsis

#include <ppcLib.h>

unsigned long ppcMfevpr(void);

Library

ppcLib.a

Description

ppcMfevpr() returns the value of the exception vector prefix register (EVPR). Bits 0 to 15 contain the prefix of the address of the exception processing routines. Bits 15 to 31 are reserved.

Errors

None.

Example

- Get the EVPR value.

```
evpr_value= ppcMfevpr();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

Synopsis

```
#include <ppcLib.h>
unsigned long ppcMfgpr1(void);
```

Library

ppcLib.a

Description

ppcMfgpr1() returns the current value of GPR(1).

Typically, this is the value of the current stack frame.

Errors

None.

Example

See **ppcMfgpr2()**, p. 10-68.

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

ppcMfgpr2()

Synopsis

#include <ppcLib.h>

unsigned long ppcMfgpr2(void)

Library

ppcLib.a

Description

ppcMfgpr2() returns the current value of GPR(2).

For XCOFF-based OS Open this is typically the value of the table of contents (TOC) pointer for the current execution context.

Errors

None.

Example

- Retrieve TOC and stack frame base from current context.

```
toc = ppcMfgpr2();
unsigned long stack_base = ppcMfgpr1();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

unsigned long iac1_value = ppcMfiac1(void);

Library

ppcLib.a

Description

ppcMfiac1() returns the value of the instruction address compare register 1 (IAC1). The IAC1 contains the address of the instruction that the debug event will be based on. The IA1 field of the Debug Control Register (DBCR) controls the instruction address 1 debug event. Bits 30 and 31 of the IAC1 are reserved, since the address must be word aligned.

Errors

None.

Example

- Get the IAC1 register value.

```
iac1_value =ppcMfiac1();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

ppcMficcr()

Synopsis

#include <ppcLib.h>

unsigned long ppcMticcr(void);

Library

ppcLib.a

Description

ppcMticcr() returns the value of the Instruction Cache Cacheability Register (ICCR).

Errors

None.

Example

- Get the ICCR value.

```
unsigned long iccr_value=ppcMficcr();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

unsigned long = ppcMficdbdr(void);

Library

ppcLib.a

Description

ppcMficdbdr() returns the current value of the Instruction Cache Debug Data Register (ICDBDR).

<ppcLib.h> has constants defined for use with the ICDBDR register.

Errors

None.

Example

- Retrieve the value of the ICDBDR.

```
unsigned long current_icdbdr = ppcMficdbdr();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

ppcMfiocr()

Synopsis

#include <ppcLib.h>

unsigned long ppcMiocr(void)

Library

ppcLib.a

Description

ppcMfiocr() returns the current value of the Input/Output Configuration Register (IOCR). The file **<ppcLib.h>** contains several constants that can be used when accessing the IOCR.

Errors

None.

Example

- Retrieve IOCR value.

```
unsigned long iocr_value;
iocr_value=ppcMfiocr();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

unsigned long ppcMfmsr(void);

Library

ppcLib.a

Description

ppcMfmsr() returns the value of the Machine State Register(MSR).

Refer to the **<ppc_arch.h>** file for the defines of constants that can be used as masks with the MSR value.

Errors

None.

Example

See **ppcMtmsr()**, p. 10-98.

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

ppcMfpit()

Synopsis

#include <ppcLib.h>

unsigned long ppcMfpit(void);

Library

ppcLib.a

Description

ppcMfpit() returns the value of the Programmable Interval Timer (PIT).

Errors

None.

Example

- Get the current PIT value.

```
unsigned long pit_value= ppcMfpit();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

unsigned long ppcMfpmcr0(void);

Library

ppcLib.a

Description

ppcMfpmcr0() returns the value of the Power Management Control Register (PMCR0).

Errors

None.

Example

- Get the current PMCR0 value.

```
unsigned long pmcr0_value= ppcMfpmcr0();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

ppcMfpvr()

Synopsis

#include <ppcLib.h>

unsigned long ppcMfpvr(void);

Library

ppcLib.a

Description

ppcMfpvr() returns the value of the processor version register, which indicates the version and revision of the PowerPC processor.

Errors

None.

Example

- Retrieve the current value of the processor version register. Processor version-specific code may require this value.

```
printf("This is processor version %x\n", ppcMfpvr());
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

unsigned long ppcMfsgr(void);

Library

ppcLib.a

Description

ppcMfsgr() returns the value of the Storage Guarded Register (SGR).

Errors

None.

Example

- Retrieve the current value of the SGR.

```
unsigned long current_sgr=ppcMfsgr();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

ppcMfsler()

Synopsis

#include <ppcLib.h>

unsigned long ppcMfsler(void);

Library

ppcLib.a

Description

ppcMfsler() returns the value of the Storage Little Endian Register (SLER).

Errors

None.

Example

- Retrieve the current value of the SLER.

```
unsigned long current_sler = ppcMfsler();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

unsigned long ppcMfsprg0(void);

unsigned long ppcMfsprg1(void);

unsigned long ppcMfsprg2(void);

unsigned long ppcMfsprg3(void);

Library

ppcLib.a

Description

ppcMfsprg0() - ppcMfsprg3() returns the current value of the special purpose register generals (SPRG0 - SPRG3).

Typically, the SPRGs provide temporary storage at the operating system level.

NOTE: OS Open reserves these registers for its own use.

Errors

None.

Example

- Read value of SPRG0.

```
unsigned long sprg0_value = ppcMfsprg0();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

ppcMfsrr0()

Synopsis

#include <ppcLib.h>

unsigned long ppcMfsrr0(void);

Library

ppcLib.a

Description

ppcMfsrr0() returns the value of SRR0.

Typically, SRR0 is used in interrupt handlers, as it usually contains the address of the next instruction to be executed at the time of the interrupt.SRR0 and SRR1 are set for protection, external, alignment, program, PIT, FIT, and syscall interrupts.

Errors

None.

Example

- Retrieve the current value of the SRR0. An exception handler may use this value to determine the point of exception.

```
unsigned long current_srr0=ppcMfsrr0();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- ppcMfsrr1(), p. 10-81
- *PPC401GF Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

unsigned long ppcMfsrr1(void);

Library

ppcLib.a

Description

ppcMfsrr1() returns the current value of SRR1.

Typically, SRR1 is used in interrupt handlers, as it contains the old MSR value as well as information bits specific to the interrupt. The file**<ppcLib.h>** contains several constants that can be used when setting the MSR values in the SRR1 register.

Errors

None.

Example

- Retrieve the current value of SRR1. This register contains the saved MSR, which may be needed by an exception handler.

```
unsigned long current_srr1=ppcMfsrr1();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

ppcMfsrr2()

Synopsis

#include <ppcLib.h>

unsigned long ppcMfsrr2(void);

Library

ppcLib.a

Description

ppcMfsrr2() returns the current value of SRR2.

Typically, SRR2 is used in interrupt handlers, as it contains the address of the next instruction which was to be executed next at the time the exception occurred. SRR2 and SRR3 are set for critical, machine check, watchdog, and debug interrupts.

Errors

None.

Example

- Retrieve the current value of SRR2. This register contains the address of the instruction that was to be executed next, which may be needed by an exception handler.

```
unsigned long current_srr2=ppcMfsrr2();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *ppcMfsrr3(), p. 10-83*
- *PPC401GF Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

unsigned long ppcMfsrr3(void);

Library

ppcLib.a

Description

ppcMfsrr3() returns the current value of SRR3.

Typically, SRR3 is used in the critical interrupt handler, as it contains the old MSR value as well as information bits specific to the interrupt. The file **<ppcLib.h>** contains several constants that can be used when setting the MSR values in the SRR3 register.

Errors

None.

Example

- Retrieve the current value of SRR3. This register contains the saved MSR, which may be needed by an exception handler.

```
unsigned long current_srr3=ppcMfsrr3();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

ppcMftb()

Synopsis

#include <ppcLib.h>

void ppcMftb(tb_t *clock_data);

Library

ppcLib.a

Description

ppcMftb() returns the current time base data.

Typically, the time base registers are used to determine the number of clock cycles that have passed.

Errors

None.

Example

- Retrieve the current value of time base high and low registers.

```
tb_t clock_data;
ppcMftb(&clock_data);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

unsigned long ppcMftsr(void);

Library

ppcLib.a

Description

ppcMftsr() returns the current value of the Timer Status Register (TSR). The file **<ppcLib.h>** contains several defined constants for the TSR that can be used as masks.

Errors

None.

Example

- Retrieve the current value of the TSR.

```
unsigned long tsr_value;
tsr_value = ppcMftsr();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

ppcMtbrcr0() • ppcMtbrcr7()

Synopsis

#include <ppcLib.h>

void ppcMtbrcr0(unsigned long brcr0_value);

void ppcMtbrcr1(unsigned long brcr1_value);

void ppcMtbrcr2(unsigned long brcr2_value);

void ppcMtbrcr3(unsigned long brcr3_value);

void ppcMtbrcr4(unsigned long brcr4_value);

void ppcMtbrcr5(unsigned long brcr5_value);

void ppcMtbrcr6(unsigned long brcr6_value);

void ppcMtbrcr7(unsigned long brcr7_value);

Library

ppcLib.a

Description

ppcMtbrcr0() - ppcMtbrcr7() set the respective Bus Region Control Register with the specified value. The file **<ppcLIb.h>** contains several constants that can be used when modifying the BRCR registers.

Errors

None.

Example

- Set the BRCR4 with a bus width of 32 bits.

```
unsigned long brcr4_value = ppcMfbrcr4();
brcr4_value = brcr4_value & 0xFFFF FFFC;
ppcMtbr4(brcr4_value | BRCR_BW_32);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

void ppcMtcdbcr(unsigned long cdbcr_value);

Library

ppcLib.a

Description

ppcMtcdbcr() sets the CDBCR to the specified value.

<ppcLib.h> has constants defined for use with the Cache Debug Control Register (CDBCR) registers.

Errors

None.

Example

- Set value of the CDBCR.

```
#include<ppcLib.h>
ppcMtcdbcr(CDBCR_CIS);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

ppcMtdac1()

Synopsis

#include <ppcLib.h>

void ppcMfdac1(unsigned long dac1_value);

Library

ppcLib.a

Description

ppMfdac1() sets the value of the appropriate Data Address Compare register. the DAC1 register contains addresses for which debug events may be taken, depending on the values set in the DBCR.

Errors

None.

Example

- Set the value of DAC1to address 0x0.

```
ppcMfdac1(0x0);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

void ppcMtdbcr(unsigned long dbcr_value);

Library

ppcLib.a

Description

ppcMtdbcr() sets the value of the debug control register (DBCR) to the specified value. The DBCR is used to enable debug events, reset the processor, control timer operations during debug events, and set the debug mode of the processor.

WARNING: Enabling bits 0 and 1 can cause unexpected results. Enabling bits 2 and 3 will cause a processor reset to occur. The DBCR is designed to be used by development tools, not applications.

File **<ppcLib.h>** has several defined constants for the DBCR.

Errors

None.

Example

- Enable external debug mode.

```
ppcMtdbcr(DBCR_EDM);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

ppcMtdbsr()

Synopsis

#include <ppcLib.h>

void ppcMtdbsr(unsigned long dbsr_value);

Library

ppcLib.a

Description

ppcMtdbsr() sets the value of the debug status register (DBSR) to the specified value. The DBSR contains the status of debug events, the JTAG serial buffers, and the most recent reset. Bits in the DBSR are cleared by writing a 1 to the corresponding bit position.

WARNING: The DBSR is designed to be used by development tools, not application software. It is strongly recommended that this register be treated as a read only register.

The file **<ppcLib.h>** defines constant values that can be used when setting DBSR

Errors

None.

Example

- Set the system reset bits.

```
ppcMtdbsr(DBSR_MRR_SYS);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

unsigned long ppcMtdccr(void);

Library

ppcLib.a

Description

ppcMtdccr() sets the value of Data Cache Cacheability Register (DCCR).

Errors

None.

Example

- Set the value of the DCCR so all regions are cacheable.

```
#include<ppcLib.h>
ppcMfdccr(0xffffffff);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

ppcMtdcwr()

Synopsis

#include <ppcLib.h>

void ppcMtdcwr(unsigned long dcwr_value);

Library

ppcLib.a

Description

ppcMtdcwr() sets the Data Cache Write-thru Register (DCWR) to the specified value.

Errors

None.

Example

- Set the value of the DCWR.

```
#include<ppcLib.h>
ppcMtdcwr(0x80000000);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

void ppcMtesr(unsigned long esr_value);

Library

ppcLib.a

Description

ppcMtesr() sets the value of the Exception Syndrome Register (ESR) to the specified value. Bits 7 to 31 are reserved.

Errors

None.

Example

- Set the all exception s off.

```
ppcMtesr(0x0);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

ppcMtevpr()

Synopsis

#include <ppcLib.h>

void ppcMtevpr(unsigned long evpr_value);

Library

ppcLib.a

Description

ppcMtevpr() sets the value of the exception vector prefix register (EVPR). Bits 0 to 15 contain the prefix of the address of the exception processing routines. Bits 15 to 31 are reserved.

WARNING: Do not use **ppcMtevpr()** if using OS Open services that use interrupts, ethernet, or SL/IP etc...

Errors

None.

Example

- Set the EVPR to 0x00A00000.

```
ppcMtevpr(0x00A00000);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

void ppcMtiac1(unsigned long iac1_value);

Library

ppcLib.a

Description

ppcMtiac1r() sets the value of the instruction address compare register 1 (IAC1). The IAC1 contains the address of the instruction that the debug event will be based on. The IA1 field of the Debug Control Register (DBCR) controls the instruction address 1 debug event. Bits 30 and 31 of the IAC1 are reserved, since the address must be word aligned.

Errors

None.

Example

- Set the IAC1 register to 0x1000.

```
ppcMtiac1(0x00001000);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

ppcMticcr()

Synopsis

#include <ppcLib.h>

void ppcMticcr(unsigned long iccr_value);

Library

ppcLib.a

Description

ppcMticcr() sets the value of the instruction cache cacheability register (ICCR) to the specified value.

Errors

None.

Example

- Set the ICCR register to 0's, making no regions of memory cacheable.

```
ppcMticcr(0x00000000);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

void ppcMtiocr(unsigned long iocr_value);

Library

ppcLib.a

Description

ppcMtiocr() sets the input/output configuration register (IOCR) to the specified value. **ppcMtiocr()** allows the user to program some of the external multifunctional pins in the PPC401GF processor. The file **<ppcLib.h>** contains several constants that can be used when accessing the IOCR.

Errors

None.

Example

- Allow external interrupt 0 triggering to be edge triggered.

```
ppcMtiocr(IOCR_E0T_EDGE);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

ppcMtmsr()

Synopsis

#include <ppcLib.h>

void ppcMtmsr(unsigned long msr_value);

Library

ppcLib.a

Description

ppcMtmsr() sets the MSR to *msr_value*.

The file **<ppc_arch.h>** defines constants that can be use with the MSR.

Errors

None.

Example

- Enable external interrupts.

```
unsigned long msr = ppcMfmsr();
ppcMtmsr(msr | ppcMsrEE);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

void ppcMtpit(unsigned long pit_value);

Library

ppcLib.a

Description

ppcMtpit() sets the programmable interval timer (PIT) to the specified value.

Errors

None.

Example

- Set the PIT to a non-0 value, to cause the PIT to start decrementing.

```
ppcMtpit(0x00000001);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

ppcMtpmcr0()

Synopsis

#include <ppcLib.h>

void ppcMtpmcr0(unsigned long pmcr0_value);

Library

ppcLib.a

Description

ppcMtpmcr0() sets the Power Management Control Register to the specified value.

Errors

None.

Example

- Set the PMCR0 to 0x0.

```
ppcMtpmcr0(0x00000000);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

void ppcMtsgr(unsigned long);

Library

ppcLib.a

Description

ppcMtsgr() sets the value of the Storage Guarded Register (SGR) to the specified value.

Errors

None.

Example

- Set the value of the SGR.

```
#include <ppcLib.h>
ppcMtsgr(0x80000000);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

ppcMtsler()

Synopsis

#include <ppcLib.h>

void ppcMtsler(unsigned long);

Library

ppcLib.a

Description

ppcMtsler() sets the value of the Storage Little Endian Register (SLER) to the specified value.

Errors

None.

Example

- Set the value of the SLER so that address range 0x50000000 - 0x57FFFFFF is accessed as little endian.

```
#include <ppcLib.h>
ppcMtsler(0x00200000);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

Synopsis

```
#include <ppcLib.h>
void ppcMtsprg0(unsigned long data);
void ppcMtsprg1(unsigned long data);
void ppcMtsprg2(unsigned long data);
void ppcMtsprg3(unsigned long data);
```

Library

ppcLib.a

Description

ppcMtsprg0() - ppcMtsprg3() set the special purpose register generals (SPRG0 - SPRG3) to the specified values.

Typically, the SPRGs provide temporary storage at the operating system level.

NOTE: OS Open reserves these registers for its own use.

Errors

None.

Example

- Set SPRG0 to 0xA0000000.

```
ppcMtsprg0(0xA0000000);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

ppcMtsrr0()

Synopsis

#include <ppcLib.h>

void ppcMtsrr0(unsigned long srr0_value);

Library

ppcLib.a

Description

ppcMtsrr0() sets the SRR0 to *srr0_value*.

Errors

None.

Example

- Set the save/restore register 0 to X'DF000000'.

```
ppcMtsrr0(0xDF000000);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

void ppcMtsrr1(unsigned long srr1_value);

Library

ppcLib.a

Description

ppcMtsrr1() sets the SRR1 to *srr1_value*.

Errors

None.

Example

- Set the save/restore register 1 to X'0000BB00'.

```
ppcMtsrr1(0x0000BB00);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

ppcMtsrr2()

Synopsis

#include <ppcLib.h>

void ppcMtsrr2(unsigned long srr2_value);

Library

ppcLib.a

Description

ppcMtsrr2() sets the SRR2 to *srr2_value*.

Errors

None.

Example

Set the save/restore register 2 to X'0000BB00'.

```
ppcMtsrr2(0x0000BB00);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

void ppcMtsrr3(unsigned long srr3_value);

Library

ppcLib.a

Description

ppcMtsrr3() sets the SRR3 to *srr3_value*. The file **<ppcLib.h>** contains several constants that can be used when setting the MSR values in the SRR3 register.

Errors

None.

Example

Set the save/restore register 3 to problem state (ppcMsrPR).

```
ppcMtsrr3(ppcMsrPR);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

ppcMttb()

Synopsis

#include <ppcLib.h>

void ppcMttb(tb_t *clock_data);

Library

ppcLib.a

Description

ppcMttb() sets the current time base data.

Typically, the time base registers are used to determine the number of clock cycles that have passed.

Errors

None.

Example

- Set the current value of time base high and low registers.

```
tb_t clock_data;
ppcMttb(0x00000000);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

void ppcMttcr(unsigned long tcr_value);

Library

ppcLib.a

Description

ppcMttcr() sets the timer control register to the specified value.

The WRC bits of the TCR 3 may only be set once, and will be reset by any form of processor reset. File **<ppcLib.h>** defines several constants for the TCR that can be used as masks.

Errors

None.

Example

- Set the TCR to force a system reset.

```
ppcMttcr(TCR_WD_SYS);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

ppcMttsr()

Synopsis

#include <ppcLib.h>

void ppcMttsr(unsigned long tsr_value);

Library

ppcLib.a

Description

ppcMttsr() sets the timer status register to the specified value. Bits in the TSR may be cleared by writing a 1 to the corresponding bit position.The file **<ppcLib.h>** defines several constants for the TSR that can be used as masks.

Errors

None.

Example

Reset the watchdog interrupt status in the TSR register.

```
ppcMttsr(TSR_WIS);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

Synopsis

#include <ppcLib.h>

unsigned long ppcOrMsr(unsigned long value);

Library

ppcLib.a

Description

ppcOrMsr() performs the OR of *value* and the current MSR, updating the MSR.

The previous value of the MSR is returned.

The file **<ppcLib.h>** defines several constants for the MSR that can be used as masks.

Errors

None.

Example

- Enable debug exceptions.

```
unsigned long old_val = ppcOrMsr(ppcMsrDE);
```

Attributes

Async Safe	No
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

ppcSync()

Synopsis

#include <ppcLib.h>

void ppcSync(void);

Library

ppcLib.a

Description

ppcSync() causes the processor to wait until all data cache lines scheduled to be written to main storage have actually been written.

Errors

None.

Example

- Ensure a **ppcDbci()** completes before using the values.

```
char *memptr = (char *)0x2000;
char new_value;
ppcDcbi((void *)memptr)
ppcSync();
new_value = *memptr;
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User's Manual*

Synopsis

#include <tickLib.h>

unsigned long processor_speed(void);

Library

tickLib.a

Description

processor_speed() returns the internal clock speed of the 401 processor.

Possible return values are:
25000000 (25 MHz)
33333333 (33 MHz)
50000000 (50 MHz)
66666666 (66 MHz)

Errors

None.

Example

- Return the the internal processor clock speed.

```
#include <tickLib.h>
unsigned long proc_speed=processor_speed();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes
Callable from Application Thread Group	Yes

Processor

PowerPC 401GF	Yes

References

- *PowerPC 401GF Embedded Controller User's Manual*

s1dbprintf()

Synopsis

#include <sys/asyncLib.h>

int s1dbprintf(unsigned long uart_clock, unsigned char *base_reg, int reg_delta, event_t event, const char *format,...);

Library

asyncLib.a

Description

s1dbprintf() is a version of **printf()** that uses polled writes (no interrupts), and may be used before I/O has been established. **s1dbprintf()** may be called before the async device driver is installed. ***uart_clock*** is the clock frequency of the serial port. *base_reg* specifies the address of the base UART register. *reg_delta* specifies the space between UART registers. *event* specifies the external interrupt level. For the 401 EVB, *uart_clock* must be 1843200, *base_reg* must be 0x7E000000, *reg_delta* should be 1.

Errors

None.

Example

- Print “Hello World” before I/O has been initialized.

```
#include <sys/asyncLib.h>
#define S1DB_PARMS 1843200, (unsigned char *)
0x7e000000, 1, EXT_IRQ_COM1
s1dbprintf(S1DB_PARMS, ”Hello World\n\r”);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User’s Manual*

Synopsis

#include <sys/asyncLib.h>

int s2dbprintf(unsigned long uart_clock, unsigned char *base_reg, int reg_delta, event_t event, const char *format,...);

Library

asyncLib.a

Description

s2dbprintf() is a version of **printf()** that uses polled writes (no interrupts), and may be used before I/O has been established. **s2dbprintf()** may be called before the async device driver is installed. ***uart_clock*** is the clock frequency of the serial port. *base_reg* specifies the address of the base UART register. *reg_delta* specifies the space between UART registers. *event* specifies the external interrupt level. For the 401 EVB, *uart_clock* must be 1843200*, base_reg* must be 0x7E000080, *reg_delta* should be 1.

Errors

None.

Example

- Print “Hello World” before I/O has been initialized.

```
#include <sys/asyncLib.h>
#define S2DB_PARMS 1843200, (unsigned char *)
0x7e000080, 1, EXT_IRQ_COM2
s2dbprintf(S2DB_PARMS, ”Hello World\n\r”);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User’s Manual*

timertick_install()

Synopsis

#include <tickLib.h>

int timertick_install(void);

Library

tickLib.a

Description

timertick_install() installs and starts the timer tick handler to maintain time-of-day in the OS Open real-time executive.

Errors

[ENOMEM] Insufficient memory to install the timer tick handler.

Example

- Do a **timertick_install()** for a 401GF processor.

```
timertick_install();
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- timertick_remove(), p. 10-117
- ppcMfpmcr0(), p. 10-75

Synopsis

#include <tickLib.h>

int timertick_remove(void);

Library

tickLib.a

Description

timertick_remove() removes the timer tick handler installed by **timertick_install()**.

Errors

[EINVAL]	Internal error involving tick handler level.

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- timertick_install(), p. 10-116

vs1dbprintf()

Synopsis

#include <sys/asyncLib.h>

int vs1dbprintf(unsigned long uart_clock, unsigned char *base_reg, int reg_delta, event_t event,const char *format,va_list arg_list);

Library

asyncLib.a

Description

vs1dbprintf() is a version of **printf()** that uses polled writes (no interrupts), and may be used before I/O has been established and accepts a va_list as a parameter instead of a variable number of parameters. **vs1dbprintf()** may be called before the async device driver is installed. ***uart_clock*** is the clock frequency of the serial port. *base_reg* specifies the address of the base UART register. *reg_delta* specifies the space between UART registers. *event* specifies the external interrupt level. For the 401 EVB, *uart_clock* must be 1843200*, base_reg* must be 0x7E000000, *reg_delta* should be 1.

Errors

None.

Example

- Print “Hello World” before I/O has been initialized.

```
#include <sys/asyncLib.h>
#define S1DB_PARMS 1843200, (unsigned char *)
0x7E000000,1, EXT_IRQ_COM1
vs1dbprintf(S1DB_PARMS, ”Hello World\n\r”);
```

Attributes

Async Safe	Yes
Cancel Safe	Yes
Interrupt Handler Safe	Yes

References

- *PPC401GF Embedded Controller User’s Manual*

A

Program Trace Calls

This appendix describes the remote debugging interface provided by the ROM monitor. These calls may be used by remote debuggers other than the **RISCWatch** debugger provided with the 401 EVB kit.

A.1 Overview

The following section describes the message (ptrace) protocol that has been implemented in the ROM monitor to support debug. If you want to interface your own debugger to the ROM monitor or modify the ROM monitor to interface with your debugger, you will need to understand the existing message protocol associated with the various debugging functions.

The ptrace interface to the ROM monitor can best be understood by reviewing the information below along with the debug-specific ROM monitor source code (dbLib/ptrace.c).

A.2 MSGDATA Structure

In the interface descriptions shown below, several references are made to a “process id.” The concept of process ids does not apply to the ROM monitor, so any nonzero value can be used. The ROM monitor uses the value “42”.

Data structure "MSGDATA" is defined in dbg.h. New register definitions and new error messages are also defined in dbg.h file.

dbg.h File

```
/* @(#)dbg.h  4.3 5/9/95 09:12:14 */
/*--------------------------------------------------------------------+
|       COPYRIGHT   I B M   CORPORATION 1994
|       LICENSED MATERIAL  -  PROGRAM PROPERTY OF I B M
|       REFER TO COPYRIGHT INSTRUCTIONS: FORM G120-2083
|       US Government Users Restricted Rights - Use, duplication or |
disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
+--------------------------------------------------------------------*/
#if !defined(DBG_H)
#define DBG_H
#define BREAKPT 0x7D821008
```

```
#ifndef MIN
#define MIN(X,Y) ((X) < (Y) ? (X) : (Y))
#endif
/*ptrace definitions based on AIX ptrace                              */
#define RD_TRACE_ME       0     /* used ONLY by target task to be traced*/
#define RD_READ_I         1     /* read target instruction addr space   */
#define RD_READ_D         2     /* read target data address space       */
#define RD_READ_U         3     /* read offset from the user structure  */
#define RD_WRITE_I        4     /* write target instruction addr space  */
#define RD_WRITE_D        5     /* write target data address space      */
#define RD_WRITE_U        6     /* write offset to the user structure   */
#define RD_CONTINUE       7     /* continue execution                   */
#define RD_KILL           8     /* terminate execution                  */
#define RD_STEP           9     /**execute one or more instructions*** !*/
#define RD_READ_GPR      11     /* read general purpose register        */
#define RD_READ_FPR      12     /* read floating point register         */
#define RD_WRITE_GPR     14     /* write general purpose register       */
#define RD_WRITE_FPR     15     /* write floating point register        */
#define RD_READ_BLOCK    17     /* read block of data                   */
#define RD_WRITE_BLOCK   19     /* write block of data                  */
#define RD_ATTACH        30     /* attach to a process                  */
#define RD_DETACH        31     /* detach a proc to let it keep running */
#define RD_REGSET        32     /* return entire register set to caller */
#define RD_REATT         33     /* reattach debugger to proc            */
#define RD_LDINFO        34     /* return loaded program info           */
#define RD_MULTI         35     /* set/clear multi-processing           */
#define RD_READ_I_MULT   70     /* Read multiple inst words             */
#define RD_READ_GPR_MULT 71     /* Read multiple registers              */
#define RD_SINGLE_STEP  100     /**source line single step************ !*/
#define RD_LOAD         101     /* load a task                         !*/
#define RD_LOGIN        103     /*ptrace for login                     !*/
#define RD_LOGON        103     /*ptrace for logon                     !*/
#define RD_LOGOFF       104     /*ptrace for logoff                    !*/
#define RD_FILL         105     /*ptrace for fill memory               !*/
#define RD_PASS         106     /*ptrace for pass                      !*/
#define RD_SEARCH       107     /*ptrace for search memory             !*/
#define RD_WAIT         108     /*ptrace for wait status information   !*/
/* Added to support ADEPT */
#define RD_READ_DCR     110     /*ptrace for reading DCR's             */
#define RD_WRITE_SPR    111     /*ptrace for writing SPR's             */
#define RD_WRITE_DCR    112     /*ptrace for writing DCR's             */
#define RD_STOP_APPL    113     /*ptrace for stopping the application  */
#define RD_STATUS       114     /*ptrace for getting run status        */
#define RD_READ_SPR     115     /*ptrace for reading SPR's             */
/* Added to support 403GC */
#define RD_READ_TLB     116     /*ptrace for readingTLB(403GC )        */
#define RD_WRITE_TLB    117     /*ptrace for writing TLB(403GC )       */
/* Added to support 602 */
#define RD_READ_SR      118     /*ptrace for reading SR's              */
#define RD_WRITE_SR     119     /*ptrace for writing SR's              */
```

```
#define MAX_PTRACE      119     /*last ptrace number                    */
#define RL_LOAD_REQ     180     /* Remote Loader - Load Request         */
#define RL_LDINFO       181     /* Remote Loader - Load Information     */
/*TCP/IP services for all sorts of remote debug                         */
#define OSOPEN_SERVNAME “osopen-dbg” /* OS/Open debug service           */
#define OSOPEN_MON_SERVNAME “osopen-mon” /* OS/Open debug monitor svc   */
/*new register definition                                               */
#define DAR   137               /* Data Address Register ($dar)        */
#define DSISR 138               /* Data St Int Status Reg ($dsisr)     */
#define SRR0  139               /* Save and Restore Register 0 ($srr0) */
#define SRR1  140               /* Save and Restore Register 0 ($srr1) */
#define SR0   141               /* Segment Register ($sr0)             */
#define SR1   142               /* Segment Register ($sr1)             */
#define SR2   143               /* Segment Register ($sr2)             */
#define SR3   144               /* Segment Register ($sr3)             */
#define SR4   145               /* Segment Register ($sr4)             */
#define SR5   146               /* Segment Register ($sr5)             */
#define SR6   147               /* Segment Register ($sr6)             */
#define SR7   148               /* Segment Register ($sr7)             */
#define SR8   149               /* Segment Register ($sr8)             */
#define SR9   150               /* Segment Register ($sr9)             */
#define SR10  151               /* Segment Register ($sr10)            */
#define SR11  152               /* Segment Register ($sr11)            */
#define SR12  153               /* Segment Register ($sr12)            */
#define SR13  154               /* Segment Register ($sr13)            */
#define SR14  155               /* Segment Register ($sr14)            */
#define SR15  156               /* Segment Register ($sr15)            */
#define DEC   157               /* Decrementer ($dec)                  */
#define RTCU  158               /* Real Time Clock Upper ($rtcu)       */
#define RTCL  159               /* Real Time Clock Lower ($rtcl)       */
#define SDR0  160               /* Storage Description Reg ($sdr0)     */
#define SDR1  161               /* Storage Description Reg ($sdr1)     */
#define EIS0  162               /* External Int Summary Reg1($eis1)    */
#define EIS1  163               /* External Int Summary Reg2($eis2)    */
#define EIM0  164               /* External Int Mask Reg1($eim1)       */
#define EIM1  165               /* External Int Mask Reg2($eim2)       */
#define SRR2  166               /* Save and Restore Register 2 ($srr2) */
#define SRR3  167               /* Save and Restore Register 3 ($srr3) */
/*other definitions needed for remote debug                             */
#define RD_MAXDATA   1800        /* Total no of DWORDS in a MSGDATA    */
#define RD_MINLENGTH 6          /* Min no of dwords in msg             */
#define RD_MINBYTES (RD_MINLENGTH*sizeof(unsigned long))
#define RD_MAXBUFFER (RD_MAXDATA - RD_MINLENGTH)
#define RD_MAXPACKET 1000000    /* Max bytes in TCP/IP packet          */
#define RD_REGBYTES  (32+8)*4   /* No of bytes for all registers       */
#define NO_KILL            1    /*do not kill any users processes      */
#define KILL_PROC          0    /*kill user process upon logoff        */
#define MAX_ERROR       1014    /*last error for rptrace               */
#define MIN_ERROR       1000    /*first error for rptrace              */
#define MIN_PACKET_SIZE   24
```

```
#define DBG_SPORT   20044
#define DBG_DPORT   20050
/*new error codes                                                    */
#define RD_NOLOAD_ERR   1000    /*no loader info available            */
#define RD_COM_ERR      1001    /*communication error occured         */
#define RD_SIZE_ERR     1002    /*not enough room to pass all info   */
#define RD_NOTSUPP      1003    /*call not supported                  */
#define RD_REG_ERR      1004    /*invalid register number requested   */
#define RD_NOTAVAIL     1005    /*call not implemented at this time   */
#define RD_NOFILE_ERR   1006    /*file could not be loaded, no file   */
#define RD_NOSCAN_ERR   1008    /*could not locate scan string file   */
#define RD_NOPERM       1010    /*no permission to log on             */
#define RD_INVALID_SEQ  1011    /*invalid rptrace sequence            */
#define RD_BUSY_ERR     1012    /*some users is already logged on      */
#define RD_PTRACE_ERR   1014    /*internal ptrace error               */
#define RD_OK              0    /*rptrace completed ok                */
#define ARCH_403              0x34000000            /* 403 architecture */
#define ARCH_601              0x36000000            /* 601 architecture */
#define ARCH_602              0x36303200            /* 602 architecture */
#define ARCH_603              0x36303300            /* 603 architecture */
#define ARCH_604              0x36303400            /* 604 architecture */
typedef struct msgdata          /* message data structure              */
{   unsigned long data_len;     /* optional data length    }          */
unsigned long retcode;          /* return code             }MIN       */
unsigned long request;          /* request type            }PART      */
unsigned long address;          /*   function parameter    }=         */
unsigned long data              /*   function parameter    }6*DWORD */
struct {      unsigned f1:1;
              unsigned f2:1;
              unsigned f3:1;
              unsigned padd:21;
              unsigned f25:8;
   } flags;
#define printmsg flags.f1
#define breakpt flags.f2
#define dbg_seqno flags.f25
union {      unsigned long trace_buffer[RD_MAXBUFFER];
             unsigned long processid;
   } parameter;
#define buffer parameter.trace_buffer  /* buffer for data, in any     */
#define rpid parameter.processid       /* process id                  */
} MSGDATA;
#endif
```

A.3 Ptrace Definitions

The following section presents the application programming interface (API) for rptrace messages. One field that is not shown here, because it is common to every call, is the *msg.printmsg* flag. This may be set in an rptrace response where *msg.retcode* does not equal

RD_OK. When the *msg.printmsg* flag is set it indicates that a text string is contained in *msg.buffer* and that this message should be displayed to the user. Typically this is an error message that provides more detail as to why the rptrace call failed to return RD_OK.

Another field that is not shown is the *dbg_seqno* field. The field provides a mechanism for recovering from lost requests and responses. If a request has the *dbg_seqno* field as not zero, it is compared with the value from the previous request. If it matches, the action is not performed and instead, the previous response is sent. This allows the debugger to time-out and re-try requests without danger of performing the same function twice.

A.3.1 RD_ATTACH (30)

Attaches debugger to running process in target environment.

A.3.1.1 Request data

Table A-1. RD_ATTACH Request Table

Parameters	Description
msg.request= RD_ATTACH	Requested API function.
msg.rpid= process_id	Numeric process ID on the target system.(Any non zero value)
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

A.3.1.2 Response data

Table A-2. RD_ATTACH Response Table

Parameters	Description
msg.retcode= ESRCH (3)	The *msg.pid* parameter identifies a process that does not exist.
msg.retcode= EIO (5)	One of the parameters is incorrect.
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_NOTSUPP (1003)	Call not supported for this interface.
msg.retcode= RD_OK (0)	Successful completion.
msg.data_len=0	No additional data

A.3.2 RD_CONTINUE (7)

This request causes the process to resume execution. If the *dbg_seqno* field of the request is zero, the response is not returned until the process stops due to a breakpoint or error. Otherwise, an immediate response is sent from the **RD_CONTINUE** request and the debugger should send the **RD_STATUS** request to see if the process has stopped.

A.3.2.1 Request data

Table A-3. RD_CONTINUE Request Table

Parameters	Description
msg.request= RD_CONTINUE	Requested API function.
msg.address= address	This field is ignored by ROM monitor.
msg.data= signal	0
msg.rpid= process_id	Numeric process ID on the target system.
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

A.3.2.2 Response data

Table A-4. RD_CONTINUE Response Table

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.data= 0	

A.3.3 RD_DETACH (31)

Detaches debugger from running process in target environment. Debugged process is restarted and execution continues without debugger control.

A.3.3.1 Request data

Table A-5. RD_DETACH Request Table

Parameters	Description
msg.request= RD_DETACH	Requested API function.
msg.rpid= process_id	Numeric process ID on the target system.
msg.data= 0	Ignored by ROM monitor.
msg.address=1	Ignored by ROM monitor.
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

A.3.3.2 Response data

Table A-6. RD_DETACH Response Table

Parameters	Description
msg.retcode= ESRCH (3)	The *msg.pid* parameter identifies a process that does not exist, or a process that is currently not being debugged.
msg.retcode= RD_COM_ERR (1001)	Communications error occurred.
msg.retcode= RD_NOTSUPP (1003)	Call not supported for this interface.
msg.retcode= RD_OK (0)	Successful completion.
msg.retcode= EIO (5)	One of the parameters is incorrect.
msg.data_len= 0	No additional data is being sent.

A.3.4 RD_FILL (105)

Fills memory with zeroes at the location specified by *address* for the number of bytes specified by data.

A.3.4.1 Request data

Table A-7. RD_FILL Request Table

Parameters	Description
msg.request= RD_FILL	Requested API function.
msg.rpid= process_id	Numeric process ID on the target system.
msg.address= address	Address of memory to fill with zeroes
msg.data= count	Number of bytes to fill with zeroes
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

A.3.4.2 Response data

Table A-8. RD_FILL Response Table

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communications error occurred.
msg.retcode= RD_NOTSUPP (1003)	Call not supported for this interface.
msg.retcode= RD_OK (0)	Successful completion.
msg.retcode= EIO (5)	One of the parameters is incorrect.
msg.data_len= 0	No additional data is being sent.

A.3.5 RD_KILL (8)

This request causes the process to terminate the same way it would with an exit routine. The ROM monitor does not implement this function but simply returns an **RD_OK** response for compatibility with older debuggers.

A.3.5.1 Request data

Table A-9. RD_KILL Request Table

Parameters	Description
msg.request= RD_KILL	Requested API function.
msg.rpid= process_id	Process ID of the process to be killed.
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

A.3.5.2 Response data

Table A-10. RD_KILL Response Table

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.retcode= ESRCH (3)	The *msg.pid* parameter identifies a process that does not exist.
msg.data_len= 0	Length of additional data being sent.

A.3.6 RD_LDINFO (34)

Request loader information from target environment. This information is provided to the ROM monitor in the boot header or by the **RL_LDINFO** request. Refer to **ROM Monitor Load Format** section for more information.

A.3.6.1 Request data

Table A-11. RD_LDINFO Request Table

Parameters	Description
msg.request= RD_LDINFO	Requested API function.
msg.rpid= process_id	Process ID from which the loader information is requested.
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

A.3.6.2 Response data

Table A-12. RD_LDINFO Response Table

Parameters	Description
msg.retcode= RD_NOLOAD_ERR (1000)	No loader information is available.
msg.retcode= ESRCH (3)	The *msg.pid* parameter identifies a process that does not exist.
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_SIZE_ERR (1002)	Not enough room in the buffer to fit all load information.
msg.retcode= RD_OK (0)	Successful completion.
msg.retcode= EIO (5)	One of the parameters is incorrect.
msg.buffer[0]= ldinfo_next	Offset to next loader information segment. See note below.
msg.buffer[1]= fd	File descriptor for loaded object. In remote debug 0xFFFF FFFF should be returned (this is a space filler).

Table A-12. RD_LDINFO Response Table

Parameters	Description
msg.buffer[2]= textorig	Starting text address.
msg.buffer[3]= textsize	Size of text.
msg.buffer[4]= dataorig	Starting data address.
msg.buffer[5]= datasize	Size of data.
msg.buffer[6]= (char *)pathname	Fully qualified filename of the object file.
msg.buffer[X]= (char *)membername	Member name (used for shared library objects). **X** does not represent position on word boundary. A NULL has to be returned for the membername even if the debugged file has no membername.
msg.buffer[ldinfo_next]= ldinfo_next	Next loader block (notice "ldinfo_next").
msg.data_len= "variable"	Set to length of data sent in msg.buffer. Data length will vary depending on the amount of information passed. Remember to count all the NULL characters.
Note: *ldinfo_next=0* indicates that no further loader blocks are present, otherwise ldinfo_next contains the offset of the next loader block in the buffer. This is actually the length of the current block. For example, if the buffer contains three blocks of lengths 38, 40 and 41 bytes, the ldinfo_next fields would be 38, 40 and 0, respectively. Note also that the blocks do not have to be contiguous - it is possible that the end of one block may not directly abut the following block. This may occur if additional information or word-aligning padding is placed after the end of the membername string. Path-name and member-name are strings terminated with a NULL character.	

A.3.7 RD_LOAD (101)

Loads executable program. Full path name of the file to be loaded is passed in this message. The ROM monitor will respond by sending an **RL_LOAD_REQ** to the remote loader daemon port.

A.3.7.1 Request data

Table A-13. RD_LOAD Request Table

Parameters	Description
msg.request= RD_LOAD	Requested API function.
msg.buffer= filename	Name of file to load. A NULL character terminates filename. Filename contains fully qualified path to that file.
msg.data_len= strlen(filename)+1	String length of filename plus NULL character.

A.3.7.2 Response data

Table A-14. RD_LOAD Response Table

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.retcode= RD_NOFILE_ERR (1006)	Could not locate/load the file.
msg.rpid= process_id	Process_id of the newly loaded file. This number (integer) can not be equal to -1 (0xFFFF FFFF) or 0.
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

A.3.8 RD_LOGIN (103)

Initializes users LOGIN. This request must be the first rptrace request issued by the debugger or results will be unpredictable.

A.3.8.1 Request data

Table A-15. RD_LOGIN Request Table

Parameters	Description
msg.request= RD_LOGIN	Requested API function.
msg.buffer[0]= host_name	This field is ignored by ROM monitor.
msg.buffer[strlen(host_name)+1]= user_name	This field is ignored by ROM monitor.
msg.data_len= strlen(host_name)+strlen(user_name)+2	Length of additional data being sent.

A.3.8.2 Response data

Table A-16. RD_LOGIN Response Table

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.data_len= 0	Length of additional data being sent.

A.3.9 RD_LOGOFF (104)

Performs user LOGOFF function. This is used when the debugger performs normal termination using quit or detach.

A.3.9.1 Request data

Table A-17. RD_LOGOFF Request Table

Parameters	Description
msg.request= RD_LOGOFF	Requested API function.
msg.data= NO_KILL	This field is ignored by ROM monitor.
msg.data_len= 0	Length of additional data being sent.

A.3.9.2 Response data

Table A-18. RD_LOGOFF Response Table

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.retcode= RD_INVALID_SEQ (1011)	Not logged on.
msg.data_len= 0	Length of additional data being sent.

A.3.10 RD_READ_D (2)

This request returns the integer in the debugged process address space at the location pointed to by the *address* parameter. If the value of *address* is not in a valid address space, unpredictable results will occur.

A.3.10.1 Request data

Table A-19. RD_READ_D Request Table

Parameters	Description
msg.request= RD_READ_D	Requested API function.
msg.address= address	Address of memory to read data from.
msg.rpid= process_id	Numeric process ID on the target system.
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

A.3.10.2 Response data

Table A-20. RD_READ_D Response Table

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.retcode= EIO (5)	Debugged process can not access given address.
msg.retcode= ESRCH (3)	The *msg.pid* parameter identifies a process that does not exist.
msg.data= data	Data read at location pointed to by address. -1 if error.
msg.data_len= 0	Length of additional data being sent.

A.3.11 RD_READ_DCR (110)

This request reads data directly from one of the DCRs (not the process's copy). All DCR registers are accessible through this message request. The sender is responsible for supplying valid DCR values, no error checking is performed on this field.

A.3.11.1 Request data

A.3.11.2 Response data

A.3.12 RD_READ_GPR (11)

This request returns the content of one of the general-purpose or special-purpose registers of the debugged process. Valid registers are defined in "dbg.h" and "sys/reg.h". Not all defined registers are supported for all environments.

A.3.12.1 Request data

Table A-21. RD_READ_GPR Request Table

Parameters	Description
msg.request= RD_READ_GPR	Requested API function.
msg.rpid= process_id	Numeric process ID on the target system.
msg.address= register	Name of the register to be read.
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

A.3.12.2 Response data

Table A-22. RD_READ_GPR Response Table

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.retcode= EIO (5)	Register is not defined.
msg.retcode= RD_REG_ERR (1004)	Unable to access given register.
msg.data= value	Value read from register. 0xFFFFFFFF if error occurred.
msg.retcode= ESRCH (3)	The *msg.pid* parameter identifies a process that does not exist.
msg.data_len= 0	Length of additional data being sent.

A.3.13 RD_READ_GPR_MULT(71)

This request returns the contents of general-purpose registers 0 to 18, inclusive, of the debugged process.

A.3.13.1 Request data

Table A-23. RD_READ_GPR_MULT Request Table

Parameters	Description
msg.request= RD_READ_GPR_MULT	Requested API function.
msg.rpid= process_id	Numeric process ID on the target system.
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

A.3.13.2 Response data

Table A-24. RD_READ_GPR_MULT Response Table

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.retcode= RD_NOTSUPP (1003)	Call not supported by this interface.
msg.retcode= RD_REG_ERR (1004)	Unable to access given register.
msg.retcode= ESRCH (3)	The *msg.pid* parameter identifies a process that does not exist.
msg.data_len= 76 (0x4C)	Length of additional data being sent.
msg.buffer[0-18]	Values read from GPR0 to GPR18. Undefined if error.

A.3.14 RD_READ_I (1)

This request returns the integer in the debugged process address space at the location pointed to by the *address* parameter. If the value of *address* is not in a valid address space, unpredictable results will occur.

A.3.14.1 Request data

Table A-25. RD_READ_I Request Table

Parameters	Description
msg.request= RD_READ_I	Requested API function.
msg.address= address	Address of memory to read data from.
msg.rpid= process_id	Numeric process ID on the target system.
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

A.3.14.2 Response data

Table A-26. RD_READ_I Response Table

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.retcode= EIO (5)	Debugged process can not access given address.
msg.retcode= ESRCH (3)	The *msg.pid* parameter identifies a process that does not exist.
msg.data= data	Data read at location pointed to by address. -1 if error (retcode should also be set to EIO).
msg.data_len= 0	Length of additional data being sent.

A.3.15 RD_READ_I_MULT (71)

This request returns the 32 integers in the debugged process address space at the location pointed to by the *address* parameter. If the value of *address* is not in a valid address space, unpredictable results will occur.

A.3.15.1 Request data

Table A-27. RD_READ_I_MULT Request Table

Parameters	Description
msg.request= RD_READ_I_MULT	Requested API function.
msg.address= address	Address of memory to read data from.
msg.rpid= process_id	Numeric process ID on the target system.
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

A.3.15.2 Response data

Table A-28. RD_READ_I_MULT Response Table

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.retcode= EIO (5)	Debugged process can not access given address.
msg.retcode= ESRCH (3)	The *msg.pid* parameter identifies a process that does not exist.
msg.retcode= RD_NOTSUPP (1003)	Call not supported by this interface.
msg.buffer[0-0x1F]	Contents of addresses from location pointed to by address to address + 0x1F.
msg.data_len= 128 (0x80)	Length of additional data being sent.

A.3.16 RD_READ_SPR (115)

This request reads data directly from one of the SPRs (not the process's copy). All SPR registers are accessible through this message request. The sender is responsible for supplying valid SPR values, no error checking is performed on this field.

A.3.16.1 Request data

Table A-29. RD_READ_SPR Request Table

Parameters	Description
msg.request= RD_READ_SPR	Requested API function.
msg.address= SPR number	SPR number to read.
msg.data_len= 0	Length of additional data being sent.

A.3.16.2 Response data

Table A-30. RD_READ_SPR Response Table

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.data= value	Value read from register.
msg.data_len= 0	Length of additional data being sent.

A.3.17 RD_STATUS (114)

This request is used to get program execution status and to determine if a previous **RD_CONTINUE** request was received.

A.3.17.1 Request data

Table A-31. RD_STATUS Request Table

Parameters	Description
msg.request= RD_STATUS	Requested API function.
msg.rpid= process_id	Numeric process ID on the target system
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

A.3.17.2 Response data

Table A-32. RD_STATUS Response Table

Parameters	Description
msg.address= execution status	Status is 1 if program is running and 0 if stopped. In the case of an error, this field will be -1 (0xFFFFFFFF).
msg.data= sequence number	Sequence number of the last RD_CONTINUE request that was received.
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.retcode= RD_ESRCH (3)	The msg.pid field identifies a process that does not exist.

A.3.18 RD_STOP_APPL (113)

This request is used to interrupt program execution.

A.3.18.1 Request data

Table A-33. RD_STOP_APPL Request Table

Parameters	Description
msg.request= RD_STOP_APPL	Requested API function.
msg.rpid= process_id	Numeric process ID on the target system
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

A.3.18.2 Response data

Table A-34. RD_STOP_APPL Response Table

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.retcode= RD_ESRCH (3)	The msg.pid field identifies a process that does not exist.

A.3.19 RD_WAIT (108)

This call allows the debugger to determine the current status of the debugged process after it is stopped. The first (least significant) byte of the process status indicates the reason for stoppage: this is always 0x7f. The second byte contains the signal number that caused the stop. Valid signals are:

AIX_SIGILL (4) - illegal instruction
AIX_SIGTRAP (5) - hit a trap instruction (breakpoint)
AIX_SIGFPE (8) - floating point error
AIX_SIGSEGV (11) - storage violation

For example after hitting a breakpoint, the status of 0x57f is returned to the debugger. After the program terminates, the first byte contains 0x00 and the rest of the status holds the program exit code. After RD_KILL call wait status of 0x57f should be returned.

A.3.19.1 Request data

Table A-35. RD_WAIT Request Table

Parameters	Description
msg.request= RD_WAIT	Requested API function.
msg.data_len= 0	Length of data in msg.buffer.

A.3.19.2 Response data

Table A-36. RD_WAIT Response Table

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.data= status	Process status.
msg.address= pid	Process id.
msg.data_len= strlen(message_string)	The ROM monitor always returns 0 in this field.
msg.buffer= message_string	Formatted message string text (NULL terminated).

A.3.20 RD_WRITE_BLOCK (19)

This request writes a block of data into the address space of the debugged process at the address pointed to by the *msg.address* field. The number of bytes to write is contained in the *msg.data* field and the data is in the *msg.buffer* field. Unpredictable results occur if the *msg.address* parameter points to a location that can not be accessed by the debugged process.

A.3.20.1 Request data

Table A-37. RD_WRITE_BLOCK Request Table

Parameters	Description
msg.request= RD_WRITE_BLOCK	Requested API function.
msg.address= address	Address of memory to write data to.
msg.data= count	Number of bytes of buffer area to be written
msg.buffer	Data to be written.
msg.data_len= count	Length of additional data being sent.

A.3.20.2 Response data

Table A-38. RD_WRITE_BLOCK Response Table

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.retcode= EIO (5)	Debugged process can not access given address.
msg.data_len= 0	Length of additional data being sent.

A.3.21 RD_WRITE_D (5)

This request writes the value of the *msg.data* parameter into the address space of the debugged process at the address pointed to by the *msg.address* parameter. Unpredictable results occur if the *msg.address* parameter points to a location that can not be accessed by the debugged process.

A.3.21.1 Request data

Table A-39. RD_WRITE_D Request Table

Parameters	Description
msg.request= RD_WRITE_D	Requested API function.
msg.address= address	Address of memory to write data to.
msg.data= data	Data to write to memory.
msg.rpid= process_id	Numeric process ID on the target system.
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

A.3.21.2 Response data

Table A-40. RD_WRITE_D Response Table

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.retcode= EIO (5)	Debugged process can not access given address.
msg.retcode= ESRCH (3)	The *msg.pid* parameter identifies a process that does not exist.
msg.data= data	Data written at location pointed to by address. -1 if error (retcode should also be set to EIO or ESRCH).
msg.data_len= 0	Length of additional data being sent.

A.3.22 RD_WRITE_DCR (112)

This request writes data directly to one of the DCRs (not the process's copy). All DCR registers are accessible through this request. The requester is responsible for supplying valid DCR values. No error checking is performed on this field.

A.3.22.1 Request data

Table A-41. RD_WRITE_DCR Request Table

Parameters	Description
msg.request= RD_WRITE_DCR	Requested API function.
msg.address= DCR number	DCR number to be written
msg.data= value	Data to write to register.
msg.data_len= 0	Length of additional data being sent.

A.3.22.2 Response data

Table A-42. RD_WRITE_DCR Response Table

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.data_len= 0	Length of additional data being sent.

A.3.23 RD_WRITE_GPR (14)

This request writes data to one of the general-purpose or special-purpose registers of the debugged process. Valid registers are defined in "dbg.h" and "sys/reg.h". Not all defined registers are supported for all environments.

A.3.23.1 Request data

Table A-43. RD_WRITE_GPR Request Table

Parameters	Description
msg.request= RD_WRITE_GPR	Requested API function.
msg.rpid= process_id	Numeric process ID on the target system.
msg.address= register	Name of the register to be written.
msg.data= value	Value to be written to the register.
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

A.3.23.2 Response data

Table A-44. RD_WRITE_GPR Response Table

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.retcode= EIO (5)	Register is not defined.
msg.retcode= RD_REG_ERR (1004)	Unable to access given register.
msg.data= value	Value written to register. 0xFFFFFFFF if error occurred.
msg.retcode= ESRCH (3)	The *msg.pid* parameter identifies a process that does not exist.
msg.data_len= 0	Length of additional data being sent.

A.3.24 RD_WRITE_I (4)

This request writes the value of the *msg.data* parameter into the address space of the debugged process at the address pointed to by the *msg.address* parameter. This request fails if the *msg.address* parameter points to a location that can not be accessed by debugged process. This call sets break points in the debugged process by writing TRAP (0x7D821008) instructions.

A.3.24.1 Request data

Table A-45. RD_WRITE_I Request Table

Parameters	Description
msg.request= RD_WRITE_I	Requested API function.
msg.rpid= process_id	Numeric process ID on the target system.
msg.address= address	Address of memory to write data to.
msg.data= data	Data to write to memory.
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

A.3.24.2 Response data

Table A-46. RD_WRITE_I Response Table

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.retcode= EIO (5)	Debugged process can not access given address.
msg.retcode= ESRCH (3)	The *msg.pid* parameter identifies a process that does not exist.
msg.data= data	Data written at location pointed to by address. -1 if error (retcode should also be set to EIO or ESRCH).
msg.data_len= 0	Length of additional data being sent.

A.3.25 RD_WRITE_SPR (112)

This request writes data directly to one of the SPRs (not the process's copy). All SPR registers are accessible through this request. The requester is responsible for supplying valid SPR values. No error checking is performed on this field.

A.3.25.1 Request data

Table A-47. RD_WRITE_SPR Request Table

Parameters	Description
msg.request= RD_WRITE_SPR	Requested API function.
msg.address= SPR number	SPR number to be written
msg.data= value	Data to write to register.
msg.data_len= 0	Length of additional data being sent.

A.3.25.2 Response data

Table A-48. RD_WRITE_SPR Response Table

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.data_len= 0	Length of additional data being sent.

A.3.26 RL_LDINFO (181)

This request provides load information from the host to the ROM monitor. This request is used when the target is loaded by a process other than the debugger. The information specified on the this request will be returned on subsequent **RD_LDINFO** requests.

A.3.26.1 Request data

Table A-49. RL_LDINFO Request Table

Parameters	Description
msg.request= RL_LDINFO	Requested API function.
msg.data_len= sizeof(struct ldinfo) + strlen(pathname)	Length of additional data being sent.
msg.buffer= load information	See description of RD_LDINFO request.

A.3.26.2 Response data

Table A-50. RL_LDINFO Response Table

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.data_len= 0	Length of additional data being sent.

A.3.27 RL_LOAD_REQ(180)

This request flows from the ROM monitor to the host when a RD_LOAD request is received. The port of the request is for the remote loader daemon (20050) to accommodate loading by a process independent from the debugger.

A.3.27.1 Request data

Table A-51. RL_LOAD_REQ Request Table

Parameters	Description
msg.request= RL_LOAD_REQ	Requested API function.
msg.buffer= filename	NULL terminated string containing fully qualified name of file to be loaded.
msg.data_len= strlen(filename)	Length of additional data being sent.

A.3.27.2 Response data

Table A-52. RL_LOAD_REQ Response Table

Parameters	Description
msg.retcode= RD_COM_ERR (1001)	Communication error occurred.
msg.retcode= RD_OK (0)	Successful completion.
msg.retcode= RD_NOFILE_ERR (1006)	Can’t open file or file is incorrect format.
msg.retcode= RD_PTRACE_ERR (1014)	Error reading file.
msg.rpid= process_id	Process ID of newly loaded file. This number (integer) can not be equal to -1 (0xFFFF FFFF) or 0.
msg.data_len= sizeof(msg.rpid)	Length of additional data being sent.

B

ROM Monitor Load Format

This appendix presents the ROM Monitor load format requirements.

B.1 Overview

The ROM Monitor load format is designed to permit the specification of multiple text and data sections. The format consists of a linked list of sections of specified types prefixed by a small boot header, *boot_block*, that specifies the initial target of the image and the entry point. The *boot_block* header is placed at the front of the image by **eimgbld** or **nimgbld**. The ROM Monitor does no relocation. It is assumed that the destination addresses for the individual sections are the same ones specified during the application's linkage. The *info_block* structure is reserved in the bootstrap program, bootlLib.s. **nimgbld** or **eimgbld** patch in the values within the *info_block* structure for bootLib to use at run time. The bootstrap program processes the sections back to front, that is, from the end of the image to the beginning. This is to avoid destructive overlap during the processing of typical images.

The sections are preceded by header blocks which identify the section types. The headers are linked together in a doubly linked list.

B.2 Section Types

There are three basic section types. Generally, they can occur in the image in any order, but are usually arranged in ascending address order. The section header block has the following format:

```
/*-------------------------------------------------------------------------------+
| Relocation block structure.
+-------------------------------------------------------------------------------*/
typedef struct rel_block {
   unsigned long       type;
   unsigned long       dest_addr;
   unsigned long       size;
   union {
       struct data_info {
              unsigned long   size_to_fill;
              unsigned long   char_to_fill;
```

```
        } data_info_str;
        struct text_info {
              unsigned long   toc_pointer;  /* used for XCOFF; not used for ELF */
              unsigned long   entry_pt;
        } text_info_str;
        unsigned long   number_symbols;
    } section_info;
    struct rel_block     *next;
    struct rel_block     *bptr;
} rel_block_t;
```

The **type** field is one of the following manifest constants:

```
#define TEXT_SECT       0x00000001
#define DATA_SECT       0x00000002
#define SYMB_SECT       0x00000004
```

The **dest_addr** specifies the target for the block, while **size** is the extent of the block, not counting the header. The bootstrap program uses this information to move the block to the destination specified at link time. **next** and **bptr** are the section header forward and backward pointers, respectively.

B.2.1 First Section

The first section is a text section. The ROM loader places the entire image at the address specified in the *boot_block* header. The entry point specified in the *boot_block* header is assumed to be a branch, followed by the first section header, *info_block*. This is to allow the bootstrap to easily gain immediate addressability to the first section block.

The format of the first section block is shown below:

```
/*-------------------------------------------------------------------------+
| First section header
+-------------------------------------------------------------------------*/
struct info_block {
        long magic_num;                         /* magic number                          */
        long text_start;                        /* addr of text section from section header  */
        long text_size;                         /* size of text section from section header   */
        long data_start;                        /* addr of data section from section header */
        long data_size;                         /* size of data section from section header */
        long elf_hdr_size;                      /* size of ELF headr                     */
        long sym_start;                         /* addr of symbol table                  */
        long num_syms;                          /* number of symbols                     */
        long toc_ptr;                           /* used for XCOFF; not used for ELF      */
        struct rel_block * next;                /* pointer to next boot section header   */
};
```

- **magic_num** is used for verification purposes and must be X’004D 5054’.
- **text_start** is the physical address value from the object text header.
- **text_size** is the size in bytes from the object text header.
- **data_start** is the physical address from the object data header.

- **data_size** is the size in bytes from the object data header.
- **elf_hdr_size** is the size of the object header. The debugger requires this information.
- **sym_start** is the address of the symbol table in storage.
- **num_syms** is the number of symbol entries.
- **next** points to the next section header.

B.2.2 Text Section

For a text section, the union **section_info** contains the structure **text_info**, specifying the entry point of the text section.

B.2.3 Data Section

For a data section, the union **section_info** contain the structure **data_info**, specifying **size_to_fill** and **char_to_fill**. These parameters are used to optionally fill a region past the size extent specified in the base rel_block with a character. It is most often used to zero bss by specifying the size of the bss in **size_to_fill** and 0x0 for **char_to_fill**.

B.2.4 Symbol Section

For symbols, the union **section_info** contains the number of symbols in the section. The data in this section consists of the symbol table from the original object file.

B.3 Boot Header

The entire image is preceded by the boot header that was added by **nimgbld** or **eimgbld**. The ROM loader uses this information to verify that it is a ROM Monitor load image, determine where to place the image, and whether to invoke the ROM Monitor debugger before transferring control to the entry point. The boot header is stripped off by the ROM Monitor loader and does not appear at the load address.

The boot header has the following format:

```
/*-------------------------------------------------------------------------+
| Boot header.
+-------------------------------------------------------------------------*/
typedef struct boot_block {
   unsigned long       magic;
   unsigned long       dest;
   unsigned long       num_512blocks;
   unsigned long       debug_flag;
   unsigned long       entry_point;
   unsigned long       reserved[3];
} boot_block_t;
```

- **magic** identifies this image as a legitimate ROM Monitor image and must have the value X’0052 504F’.
- **dest** is the target address for the image (after the boot header is stripped off).
- **num_512blocks** - Boot images are padded to a multiple of 512 byte blocks. This field specifies the number of blocks.
- **debug_flag** controls whether the ROM Monitor debugger gets control before the loaded image starts. If the value is 0x0, the image runs immediately. If 0x01, the debugger gains control as soon as the load is complete.
- **entry_point** specifies the address where the image will receive control.

C

FPGA Program Images

C.1 Peripheral Interface Controller

```
%*****************************************************************%
%******** PERIPHERAL CHIP SELECT AND CONTROL SIGNALS ************%
%*****************************************************************%
INCLUDE "74169" ;

SUBDESIGN GURU
(

%*****************************************************************%
%*******************  INPUT SIGNALS  ***************************%
%*****************************************************************%

  G_SPARE1 : INPUT ; % Spare input %
  /RESET   : INPUT ; % System Reset %
  CLK1     : INPUT ; % System Clock %
  DCLK1    : INPUT ; % Delayed System clock %
  A[30..0] : INPUT ; % Address %
  W_/R     : INPUT ; % System Read/Write %
  /BLAST   : INPUT ; % End of cycle signal %
  /ADS     : INPUT ; % System address Strobe/Start of cycle %
  /ACK     : INPUT ; % Ethernet register sync. hand shaking Signal %
  PRQ      : INPUT ; % Ethernet Remote DMA hand shaking Signal %
  D_ETCLK  : INPUT ; % Delayed Ethernet Clock %
  % Frequency Select input Signal %
  FREQ_SEL : INPUT;

  % FREQ_SEL = 0 -----> 25Mhz clock
    FREQ_SEL = 1 -----> 33Mhz clock %
```

```
 %*****************************************************************%
 %******************** OUTPUT SIGNALS  ***************************%
 %*****************************************************************%

  /OE         : OUTPUT ; % Output Enable for peripherals %
  /WE         : OUTPUT ; % Write Enable for peripherals %
  /READY      : BIDIR  ; % Ready for peripherals %
  LCD_EN      : OUTPUT ; % Enable Signal for LCD %
  /PLX_CS     : OUTPUT ; % PLX 9060 Chip Select %
  /ENET_CS    : OUTPUT ; % Ethernet chip select %
  /LATCH_CS   : OUTPUT ; % Ethernet PRQ Read Port Chip Select %
  /SER1_CS    : OUTPUT ; % Serial Port1 Chip Select %
  /SER2_CS    : OUTPUT ; % Serial Port2 Chip Select %
  /PAR_CS     : OUTPUT ; % Parallel Port Chip Select %
  /RTC_CS     : OUTPUT ; % Real Time Clock Chip Select %
  /FLASH_CS   : OUTPUT ; % 401GF Boot Flash Chip Select %
  /PCI_FLASH  : OUTPUT ; % PCI Expansion Flash ROM Chip Select %
  /SQUAL_CS   : OUTPUT ; % SQUALL Module Chip Select %
  /SRAM_CS    : OUTPUT ; % SRAM Chip Select %
  /INT1_CS    : OUTPUT ; % Interrupt Controller1(Master) Chip Select %
  /INT2_CS    : OUTPUT ; % Interrupt Controller2(Slave) Chip Select %
  /INT_ACK_CS: OUTPUT ; % Interrupt Acknowledge to INT Controller %
  /SerE2_CS   : OUTPUT ; % Serial EEPROM Chip Select %
  /245_OE     : OUTPUT ; % Output control signal for IO tranceiver
                           buffer %
  D7          : OUTPUT ; % PRQ status read bit %

)

VARIABLE

  COUNTER       : 74169;
  PRQLATCH      : DFF  ; % Ethernet PRQ Sync. to 33Mhz/25Mhz %
  /LATCH_ACK    : DFF  ; % Ethernet register R/W Ack. Sync. to
                           33Mhz/25Mhz %
  /LCD_CS       : NODE ; % LCD chip Select %
  /ENET_CS_NODE : NODE ; % Ethernet Chip select %
  /ENET_CS_FF   : DFF  ;
  /PLX_CS_FF    : DFF  ;
  /READY1       : NODE ;
  /READY2       : NODE ;
  /READY3       : NODE ;
  /READY4       : NODE ;
  /OE1          : NODE ;
  /OE3          : NODE ;
  /OE4          : NODE ;
  /WE1          : NODE ;
  /WE3          : NODE ;
```

```
 /WE4          : NODE ;

%********  STATE DECLARATION OF IO STATE MACHINES  *********%

IOSF : MACHINE  WITH STATES (
                                        IOS00 ,
                                        IOS1  ,
                                        IOS2  ,
                                        IOS3  ,
                                        IOS4  ,
                                        IOS5  ,
                                        IOS6  ,
                                        IOS7  ,
                                        IOS8
                            );

IOSL : MACHINE WITH STATES (
                                        IOS01 ,
                                        IOS9  ,
                                        IOS10 ,
                                        IOS11 ,
                                        IOS12 ,
                                        IOS13
                            );

IOSE : MACHINE WITH STATES (
                                        IOS02 ,
                                        IOS32 ,
                                        IOS34 ,
                                        IOS35 ,
                                        IOS36
                           );


IOSER : MACHINE WITH STATES (
                                        IOSER0 ,
                                        IOSER1 ,
                                        IOSER2 ,
                                        IOSER3 ,
                                        IOSER4 ,
                                        IOSER5
                              );

STCS : MACHINE OF BITS(/CS_FF)
        WITH STATES ( CS_FF0 = B"1", CS_FF1 = B"0");

BEGIN
```

```
%
 FUNCTION 74169 ( !IOS9, GND, GND, GND, CLK1, VCC, VCC, VCC, VCC)
 RETURNS ( Q0, Q1, Q2, Q3, TCN);
%

COUNTER.LDN  = !IOS9;
COUNTER.ENTN = GND;
COUNTER.ENPN = GND;
COUNTER.U/DN = GND;
COUNTER.CLK  = CLK1;
COUNTER.D[3..0]   = VCC;
!/READY = TRI((!/READY1 # !/READY2 # !/READY3 # !/READY4),
              (!/245_OE # !/ENET_CS_NODE # !/LATCH_CS));

!/OE = !/OE1 # !/OE3 # !/OE4;
!/WE = !/WE1 # !/WE3 # !/WE4;

% DFF is used to synchronise PRQ with
  33Mhz/25Mhz.  Data D7 is tristated
  with blast and chip select signal
  by assuming it is single access %
PRQLATCH.clk = GLOBAL(CLK1);
PRQLATCH.CLRN =  /RESET;
PRQLATCH.D = PRQ;

% PRQ status read bit Address (7400 0010 - 7400 0010) %
D7 = TRI(PRQLATCH, (!/BLAST & !/LATCH_CS & !A0));

% Following Eq. Synchronize Ethernet register
  sync. Acknowledge(/ACK) to 33Mhz/25Mhz %
/LATCH_ACK.clk = GLOBAL(CLK1);
/LATCH_ACK.PRN =  /RESET;
/LATCH_ACK.D = /ACK;
% PLX 9060 chip select active period control signal %
/PLX_CS_FF.clk  = /BLAST # /READY;
/PLX_CS_FF.PRN  = /RESET;
/PLX_CS_FF.CLRN = /ADS;
/PLX_CS_FF.D = VCC;


% CHIP SELECT SIGNAL GENERATION FOR ALL PERIPHERALS %
% 1000 0000 - 1007 FFFF(SRAM) %
!/SRAM_CS = !A30 & !A29 &  A28 & !A27 & !A26 & !A25 & !A24
       & !A23 & !A22 & !A21 & !A20 & !A19 & !/CS_FF;

% C000 0000 - CFFF FFFF(SQUALL) %
!/SQUAL_CS=  A30 & !A29 & !A28 & !/CS_FF;
```

```
% PLX Register read Chip Select %
!/PLX_CS = !A30 &  A29 & !A28 & !A27 & !A26 & !A25 & !A24 & !A23
      & !A22 & !A21 & !A20 & !A19 & !A18 & !A17 & !A16
          & !A15 & !A14 & !A13 & !A12 & !A11 & !A10 & !A9 &
                        !/PLX_CS_FF;

% 7400 0000 - 7400 000F(ETHERNET) %
!/ENET_CS_NODE  =  A30 &  A29 &  A28 & !A27 &  A26 & !A25 & !A24 & !A23
               & !A22 & !A21 & !A20 & !A19 & !A18 & !A17 & !A16
               & !A15 & !A14 & !A13 & !A12 & !A11 & !A10 & !A9
               & !A8  & !A7  & !A6  & !A5  & !A4 & !/CS_FF;

% 7E00 0000 - 7E00 0007(SERIAL PORT1) %
!/SER1_CS =  A30 &  A29 &  A28 &  A27 &  A26 &  A25 & !A24 & !A23 &
                        !A22
       & !A21 & !A20 & !A19 & !A18 & !A17 & !A16 & !A15 & !A14
        & !A13 & !A12 & !A11 & !A10 & !A9  & !A8  & !A7  & !A6
        & !A5  & !A4  & !A3  & !/CS_FF;

% 7E00 0080 - 7E00 0087 (SERIAL PORT2) %
!/SER2_CS =  A30 &  A29 &  A28 &  A27 &  A26 &  A25 & !A24 & !A23
       & !A22 & !A21 & !A20 & !A19 & !A18 & !A17 & !A16 & !A15
        & !A14 & !A13 & !A12 & !A11 & !A10 & !A9  & !A8  & A7
        & !A6  & !A5  & !A4  & !A3  & !/CS_FF;

% 7E00 0100 - 7E00 0103 (PARALLEL PORT) %
!/PAR_CS  =  A30 &  A29 &  A28 &  A27 &  A26 &  A25 & !A24 & !A23
            & !A22 & !A21 & !A20 & !A19 & !A18 & !A17 & !A16 & !A15
       & !A14 & !A13 & !A12 & !A11 & !A10 & !A9  &  A8  & !A7
        & !A6  & !A5  & !A4  & !A3  & !A2  & !/CS_FF;

% 7E01 0000 - 7E01 1FFF (RTC/NVRAM) %
!/RTC_CS  =   A30 &  A29 &  A28 &  A27 &  A26 &  A25 & !A24 & !A23
       & !A22 & !A21 & !A20 & !A19 & !A18 & !A17 & A16 & !A15
       & !A14 & !A13 & !/CS_FF;

% 7E00 0400 - 7E00 0401 (INTERRUPT CONTROLLER1) %
!/INT1_CS  =  A30 &  A29 &  A28 &  A27 &  A26 &  A25 & !A24 & !A23
         & !A22 & !A21 & !A20 & !A19 & !A18 & !A17 & !A16
         & !A15 & !A14 & !A13 & !A12 & !A11 &  A10 & !A9
         & !A8  & !A7  & !A6  & !A5  & !A4  & !A3  & !A2
         & !A1  & !/CS_FF;


% 7E00 0480 - 7E00 0481 (INTERRUPT CONTROLLER2) %
!/INT2_CS   =  A30 &  A29 &  A28 &  A27 &  A26 &  A25 & !A24 & !A23
          & !A22 & !A21 & !A20 & !A19 & !A18 & !A17 & !A16
          & !A15 & !A14 & !A13 & !A12 & !A11 &  A10 & !A9
```

```
          & !A8  &  A7  & !A6  & !A5  & !A4  & !A3  & !A2
          & !A1  & !/CS_FF;

% 7E00 0402 - 7E000402 ( 8259 INTERRUPT ACK.) %
!/INT_ACK_CS =  A30 &  A29 &  A28 &  A27 &  A26 &  A25 & !A24
   & !A23 & !A22 & !A21 & !A20 & !A19 & !A18 & !A17
   & !A16 & !A15 & !A14 & !A13 & !A12 & !A11 &  A10
   & !A9  & !A8  & !A7  & !A6  & !A5  & !A4  & !A3
   & !A2  &  A1  & !A0  & !/CS_FF;

% 7E00 0200 - 7E00 0200 (Serial EEPROM) %
!/SerE2_CS =  A30 &  A29 &  A28 &  A27 &  A26 &  A25 & !A24
         & !A23 & !A22 & !A21 & !A20 & !A19 & !A18 & !A17
         & !A16 & !A15 & !A14 & !A13 & !A12 & !A11 & !A10
         &  A9  & !A8  & !A7  & !A6  & !A5  & !A4  & !A3
         & !A2  &  !A1 & !A0  & !/CS_FF;

% FFF8 0000 - FFFF FFFF (BOOT FLASH) %
!/FLASH_CS =  A30 &  A29 &  A28 &  A27 &  A26 &  A25 &  A24
         &  A23 &  A22 &  A21 & A20 &  A19 & !/CS_FF;

% FFF0 0000 - FFF7 FFFF(PCI FLASH) %
!/PCI_FLASH = A30 &  A29 &  A28 &  A27 &  A26 &  A25 &  A24
         &  A23 &  A22 &  A21 & A20 & !A19 & !/CS_FF;

% 7400 0010 - 7400 0011 (EHTERNET PRQ/REMOTE LATCH) %
!/LATCH_CS=  A30 &  A29 &  A28 &  !A27 &  A26 &  !A25 &  !A24
        & !A23 & !A22 & !A21 & !A20 & !A19 & !A18 & !A17
        & !A16 & !A15 & !A14 & !A13 & !A12 & !A11 & !A10
        & !A9  & !A8  & !A7  & !A6  & !A5  &  A4  & !A3
        & !A2  & !A1  & !/CS_FF;

% 7E00 0300 - 7E00 0301 (LCD DISPLAY) %
!/LCD_CS=  A30 &  A29 &  A28 &  A27 &  A26 &  A25 &  !A24
      & !A23 & !A22 & !A21 & !A20 & !A19 & !A18 & !A17
      & !A16 & !A15 & !A14 & !A13 & !A12 & !A11 & !A10
      &  A9  &  A8  & !A7  & !A6  & !A5  & !A4  & !A3
      & !A2  & !A1  & !/CS_FF;


%************ TRANSCEIVER OUTPUT CONTROL SIGNAL ***********%
STCS.clk = GLOBAL(CLK1);
STCS.reset=!/RESET;

% To Distinguish Between Idle State and Cycle in Progress %

CASE STCS IS
  WHEN CS_FF0 =>
```

```
     IF !/ADS  THEN
    STCS = CS_FF1;
     ELSE STCS = CS_FF0;
     END IF;

  WHEN CS_FF1 =>
      IF (!/BLAST & !/READY) THEN STCS = CS_FF0;
      ELSE STCS = CS_FF1;
      END IF;

END CASE;

 !/245_OE = (!/FLASH_CS # !/PCI_FLASH # !/LCD_CS # !/SER1_CS
       # !/SER2_CS # !/PAR_CS # !/INT1_CS # !/INT2_CS
       # !/INT_ACK_CS # !/RTC_CS # !/SerE2_CS);


% Chip select synchronised to Ethernet clock %
  /ENET_CS_FF.clk = D_ETCLK;
  /ENET_CS_FF.prn = /RESET;
  /ENET_CS_FF.D   = /ENET_CS_NODE;
  /ENET_CS = /ENET_CS_FF;

% State Machine for generation of peripherals control signals  %
IOSF.clk = GLOBAL(CLK1);
IOSF.reset =  !/RESET;

CASE IOSF IS

WHEN IOS00 =>
    /READY1 = VCC;
        /OE1 = VCC;
    /WE1 = VCC;

   %  Cycle is for FLASH or PCI FLASH or RTC or 8259 goes from IOS00 to
                          IOS1 %
    IF (!/FLASH_CS # !/PCI_FLASH # !/RTC_CS # !/INT1_CS # !/INT2_CS #
                          !/INT_ACK_CS)
    THEN
     IOSF = IOS1;
    ELSE IOSF = IOS00;

    END IF;


%  State Machine for FLASH, RTC & 8259 to generate /OE, /WE & /Ready
                          Signals %
 WHEN  IOS1 =>
```

```
   % if 25Mhz skip 2 wait states i.e goes from state 1 to  state 4 %
   IF !FREQ_SEL THEN IOSF = IOS4;
   ELSE IOSF = IOS2;
   END IF;

   /READY1 = VCC;

   IF !W_/R THEN
      /OE1 = VCC;
      /WE1 = VCC;
   ELSE
      /OE1 = VCC;
      /WE1 = GND;
   END IF;

WHEN  IOS2 =>
   IOSF = IOS3;
   /READY1 = VCC;
   IF !W_/R  THEN
    /OE1 = GND;
    /WE1 = VCC;
   ELSE
    /OE1 = VCC;
    /WE1 = GND;
  END IF;

WHEN  IOS3 =>
   IOSF = IOS4;
   /READY1 = VCC;
   IF !W_/R  THEN
    /OE1 = GND;
    /WE1 = VCC;
   ELSE
    /OE1 = VCC;
    /WE1 = GND;
   END IF;

WHEN  IOS4 =>
   IOSF = IOS5;
   /READY1 = VCC;
   IF !W_/R  THEN
    /OE1 = GND;
    /WE1 = VCC;
   ELSE
    /OE1 = VCC;
    /WE1 = GND;
   END IF;
```

```
WHEN  IOS5 =>
   IOSF = IOS6;
   /READY1 = VCC;
   IF !W_/R  THEN
    /OE1 = GND;
    /WE1 = VCC;
   ELSE
    /OE1 = VCC;
    /WE1 = GND;
  END IF;

WHEN  IOS6 =>
   IOSF = IOS7;
   /READY1 = VCC;
   IF !W_/R  THEN
    /OE1 = GND;
    /WE1 = VCC;
   ELSE
    /OE1 = VCC;
    /WE1 = GND;
   END IF;

WHEN  IOS7 =>
   IOSF = IOS8;
   /READY1 = VCC;
   IF !W_/R  THEN
    /OE1 = GND;
    /WE1 = VCC;
   ELSE
    /OE1 = VCC;
    /WE1 = GND;
   END IF;

WHEN  IOS8 =>
   IF !/BLAST THEN   % Checks for end of the cycle %
     IOSF = IOS00;
   ELSE
     IOSF = IOS1;
   END IF;
   /READY1 = GND;
   IF !W_/R  THEN
    /OE1 = GND;
    /WE1 = VCC;
   ELSE
    /OE1 = VCC;
    /WE1 = VCC;
   END IF;
```

```
END CASE;

 % State Machine for LCD to generate Enable and Ready Signals

 Note : This doesn't support Burst Cycles. The limitation is that
    only single read/write possible. Burst access on LCD address
    space causes the system to hang. %

IOSL.clk = GLOBAL(CLK1);
IOSL.reset =  !/RESET;

CASE IOSL IS

WHEN IOS01 =>
  % If Cycle is for LCD goes from IOS01 to IOS9 %
     IF !/LCD_CS THEN
    IOSL = IOS9;
      ELSE IOSL = IOS01;
      END IF;
      LCD_EN = GND;
      /READY2 = VCC;
% LOAD THE COUNTER WITH '1111' %
WHEN IOS9 =>
  LCD_EN = GND;
  /READY2 = VCC;
  IOSL = IOS10;

% DECREMENT THE COUNTER FOR LCD_EN INACTIVE TIME %
WHEN IOS10 =>
  LCD_EN = GND;
  /READY2 = VCC;
  IF (!COUNTER.Q0 & !COUNTER.Q1 & !COUNTER.Q2 & !COUNTER.Q3) THEN IOSL
                          = IOS11;
  ELSE IOSL = IOS10;
  END IF;

% DECREMENT THE COUNTER FOR LCD_EN ACTIVE TIME%
WHEN IOS11 =>
  LCD_EN = VCC;
  /READY2 = VCC;
  IF (!COUNTER.Q0 & !COUNTER.Q1 & !COUNTER.Q2 & !COUNTER.Q3) THEN IOSL
                          = IOS12;
  ELSE IOSL = IOS11;
  END IF;

WHEN IOS12 =>
  IF !W_/R THEN
     /READY2 = GND;
```

```
  ELSE
     /READY2 = VCC;
  END IF;
  LCD_EN = VCC;
  IOSL = IOS13;

WHEN IOS13 =>
  IF !W_/R THEN
       /READY2 = VCC;
  ELSE
       /READY2 = GND;
  END IF;
  LCD_EN = GND;
  IOSL = IOS01;

END CASE;



% State Machine for SIO & Ethernet Controller %

IOSE.clk = GLOBAL(CLK1);
IOSE.reset =  !/RESET;

CASE IOSE IS

 WHEN IOS02 =>
    /READY3 = VCC;
    /OE3 = VCC;
    /WE3 = VCC;

    % If Cycle is for SIO or ethernet goes from IOS02 to IOS32 %
    IF (!/SER1_CS # !/SER2_CS # !/PAR_CS # !/LATCH_CS  # !/SerE2_CS)
    THEN IOSE = IOS32;
    ELSE IOSE = IOS02;
    END IF;

WHEN IOS32 =>
    /READY3 = VCC;

    % If cycle is for serial or par. ports skips 1 wait state %
    IF (!/SER1_CS # !/SER2_CS # !/PAR_CS) THEN IOSE =IOS34;

    % Responds to Ethernet Remote DMA/Serial EEPROM and
      skips 2 wait states %

    ELSIF (!/LATCH_CS # !/SerE2_CS) THEN IOSE = IOS35;
    ELSE IOSE = IOS32;
```

```
    END IF;

    IF !W_/R THEN
      /OE3 = GND;
      /WE3 = VCC;
    ELSE
           /OE3 = VCC;
      /WE3 = GND;
    END IF;

WHEN IOS34 =>
    IOSE = IOS35;
    /READY3 = VCC;
    IF !W_/R THEN
      /OE3 = GND;
      /WE3 = VCC;
    ELSE
           /OE3 = VCC;
      /WE3 = GND;
    END IF;

WHEN IOS35 =>
    /READY3 = GND;
    IF !/BLAST THEN % Checks for the end of cycle %
       IOSE = IOS02;
    ELSE
       IOSE = IOS36;
     END IF;

    IF !W_/R THEN
      /OE3 = GND;
      /WE3 = VCC;
    ELSE
           /OE3 = VCC;
      /WE3 = VCC;
    END IF;

WHEN IOS36 =>
   IOSE = IOS32;
   /READY3 = VCC;
   /OE3 = VCC;
   /WE3 = VCC;

END CASE;

IOSER.clk = GLOBAL(CLK1);
IOSER.reset =  !/RESET;
```

```
CASE IOSER IS

 WHEN IOSER0 =>
    /READY4 = VCC;
    /OE4 = VCC;
    /WE4 = VCC;

    % If Cycle is for ethernet goes from IOSER0 to IOSER1 %
    %
      Here /ENET_CS_NODE is used instead of /ENET_CS to qualify
      Ethernet cycle. The /ENET_CS is generated synchronous
      to Ethernet clock. The Ethernet clock may be different
      from system clock
    %
    IF !/ENET_CS_NODE THEN
     IOSER = IOSER1;
    ELSE IOSER = IOSER0;
    END IF;

WHEN IOSER1 =>
    /READY4 = VCC;

    % Ethernet register access, checking for reg. Sync. signal %
    IF (!/ENET_CS_NODE & !/LATCH_ACK) THEN IOSER = IOSER2;
    ELSE IOSER = IOSER1;
    END IF;

    IF !W_/R THEN
      /OE4 = GND;
      /WE4 = VCC;
    ELSE
           /OE4 = VCC;
      /WE4 = GND;
    END IF;

WHEN IOSER2 =>
    IOSER =IOSER3;
    /READY4 = VCC;
    IF !W_/R THEN
      /OE4 = GND;
      /WE4 = VCC;
    ELSE
           /OE4 = VCC;
      /WE4 = GND;
    END IF;

WHEN IOSER3 =>
    IOSER = IOSER4;
```

```
    /READY4 = VCC;
    IF !W_/R THEN
      /OE4 = GND;
      /WE4 = VCC;
    ELSE
           /OE4 = VCC;
      /WE4 = GND;
    END IF;


WHEN IOSER4 =>
    /READY4 = GND;
    IF !/BLAST THEN % Checks for the end of cycle %
       IOSER = IOSER0;
    ELSE
       IOSER = IOSER5;
     END IF;

    IF !W_/R THEN
      /OE4 = GND;
      /WE4 = VCC;
    ELSE
           /OE4 = VCC;
      /WE4 = VCC;
    END IF;

WHEN IOSER5 =>
   IOSER = IOSER1;
   /READY4 = VCC;
   /OE4 = VCC;
   /WE4 = VCC;

END CASE;

END;
```

C.2 Interleaved DRAM Controller

```
% ********************************************************** %
%  THIS INTERLEAVED DRAM CONTROLLER IS DESIGNED FOR USE WITH   %
%  60/70 NS DRAM SIMMS IN FAST PAGE MODE                       %
% ********************************************************** %

% THIS DESIGN WORKS WITH 25 OR 33 MHZ EXTERNAL BUS FREQUENCIES.
  SELECTION BETWEEN THE TWO IS DETERMINED BY MAKING SIGNAL
  25/33SW EITHER HIGH OR LOW; HIGH INDICATES 33MHZ OPERATION AND
  LOW INDICATES 25 MHZ OPERATION %

% FOR READ OPERATIONS, 0 WAIT STATE IS SUPPORTED.  FOR WRITE
  OPERATIONS, 0 WAIT STATE IS SUPPORTED FOR 401GF, AND 1 WAIT
  STATE FOR SQUALL AND PLX 9060 %

% REFRESH IS DONE AT APPROXIMATELY 7.9US INTERVALS FOR EACH BANK
  AT BOTH 25 AND 33 MHZ.  IT IS ALSO MAINTAINED AT AN ADEQUATE
  RATE IF BUS FREQUENCY DROPS DOWN TO AS LOW AS 5 MHZ %

% COMPILATION OF THE DESIGN SHOULD BE DONE WITH PUSH BACK OPTION
  ENABLED %


SUBDESIGN  MAVEN
(
% ********************************************************** %
%        PRIMARY INPUTS                                      %
%********************************************************** %

 CLK1           : INPUT ;  % 1X CLOCK %
 CLK2           : INPUT ;  % 2X CLOCK %
 A2             : INPUT ;  % ADDRESS A2 %
 A3             : INPUT ;  % ADDRESS A3 %
/RESET          : INPUT ;  % EXTERNAL RESET %
/ADS            : INPUT ;  % ADDRESS STROBE %
 EXTEND         : INPUT ;  % WAIT FOR SLOW SQUALL DEVICE READ
                           COMPLETION %
 25/33SW        : INPUT ;  % LOW =  25MHZ / HIGH = 33MHZ OPERATION %
 S5MHZ          : INPUT ;  % HIGH INDICATES 401 MEMCLK CLK HAS BEEN
                           REDUCED %
/BLAST          : INPUT ;  % BURST LAST %
 W_/R           : INPUT ;  % PROCESSOR READ/WRITE %
 A30            : INPUT ;  % ADDRESS A30 %
 A29            : INPUT ;  % ADDRESS A29 %
 A28            : INPUT ;  % ADDRESS A28 %
```

```
 A27            : INPUT ;  % ADDRESS A27 %
 A25            : INPUT ;  % ADDRESS A25 %
 A23            : INPUT ;  % ADDRESS A23 %
 A21            : INPUT ;  % ADDRESS A21 %
 PD2            : INPUT ;  % SIMM SIZE %
 PD1            : INPUT ;
 PD0            : INPUT ;
 HOLDACK        : INPUT ;  % HOLDACK SIGNAL FROM 401GF %
 DCLK1          : INPUT ;  % DELAYED 1X CLOCK %
/BE3            : INPUT ;  % BYTE ENABLE 3 %
/BE2            : INPUT ;  % BYTE ENABLE 2 %
/BE1            : INPUT ;  % BYTE ENABLE 1 %
/BE0            : INPUT ;  % BYTE ENABLE 0 %
 M_SPARE1       : INPUT ;  % SPARE INPUT %

% ********************************************************** %
%        PRIMARY OUTPUTS                                     %
% ********************************************************** %

/WRO           : OUTPUT;  % ODD BANK WE %
/WRE           : OUTPUT;  % EVEN BANK WE %
/CASEE_B0      : OUTPUT;  % BYTE 0 EVEN CAS %
/CASEE_B1      : OUTPUT;  % BYTE 1 EVEN CAS %
/CASEE_B2      : OUTPUT;  % BYTE 2 EVEN CAS %
/CASEE_B3      : OUTPUT;  % BYTE 3 EVEN CAS %
/CASOO_B0      : OUTPUT;  % BYTE O ODD CAS %
/CASOO_B1      : OUTPUT;  % BYTE 1 ODD CAS %
/CASOO_B2      : OUTPUT;  % BYTE 2 ODD CAS %
/CASOO_B3      : OUTPUT;  % BYTE 3 ODD CAS %
/ADDRMUX       : OUTPUT;  % SELECT BETWEEN ROW/COLUMN ADDRRESS %
/RASE          : OUTPUT;  % EVEN RAS %
 A3ODD         : OUTPUT;  % A3 FOR ODD BANK %
 A3EVEN        : OUTPUT;  % A3 FOR EVEN BANK %
/RASO          : OUTPUT;  % ODD RAS %
/RDY           : OUTPUT;  % READY SIGNAL TO MASTER %
/BANKSELA      : OUTPUT;  % EVEN ODD DATA SELECT FOR READ %
/BANKSELB      : OUTPUT;  % EVEN ODD DATA SELECT FOR READ %
/RDEN          : OUTPUT;  % MEMORY OUTPUT ENABLE %
/RASE_0        : OUTPUT;  % RAS0, RAS2 FOR EVEN BANK %
/RASE_1        : OUTPUT;  % RAS1, RAS3 FOR EVEN BANK %
/RASO_0        : OUTPUT;  % RAS0, RAS2 FOR ODD BANK %
/RASO_1        : OUTPUT;  % RAS1, RAS3 FOR ODD BANK %
)


VARIABLE

 AAA: DFF;
```

```
BBB: DFF;

ss: MACHINE OF BITS (A3ODD)
    WITH STATES  ( s0,s1 );
y:   NODE;
z:   NODE;

yy: MACHINE OF BITS (A3EVEN)
    WITH STATES  ( y0,y1 );
c:   NODE;
b:   NODE;

zz: MACHINE OF BITS (ADDRMUX)
    WITH STATES  ( z0,z1 );
j:   NODE;
k:   NODE;

bb: MACHINE OF BITS (BANKSELA)
    WITH STATES  ( b0,b1 );
b11: NODE;
b12: NODE;

cc: MACHINE OF BITS (BANKSELB)
    WITH STATES  ( c0,c1 );
c11: NODE;
c12: NODE;

dd: MACHINE OF BITS (CASEE_B0)
    WITH STATES  ( d0,d1 );
d11: NODE;
d12: NODE;

ee: MACHINE OF BITS (CASEE_B1)
    WITH STATES  ( e0,e1 );
e11: NODE;
e12: NODE;

ff: MACHINE OF BITS (CASEE_B2)
    WITH STATES  ( f0,f1 );
f11: NODE;
f12: NODE;

gg: MACHINE OF BITS (CASEE_B3)
    WITH STATES  ( g0,g1 );
g11: NODE;
g12: NODE;
```

```
hh: MACHINE OF BITS (CASOO_B0)
    WITH STATES  ( h0,h1 );
h11: NODE;
h12: NODE;

ii: MACHINE OF BITS (CASOO_B1)
    WITH STATES  ( i0,i1 );
i11: NODE;
i12: NODE;

jj: MACHINE OF BITS (CASOO_B2)
    WITH STATES  ( j0,j1 );
j11: NODE;
j12: NODE;

kk: MACHINE OF BITS (CASOO_B3)
    WITH STATES  ( k0,k1 );
k11: NODE;
k12: NODE;

ll: MACHINE OF BITS (CASPIPE)
    WITH STATES  ( l0,l1 );
l11: NODE;
l12: NODE;

mm: MACHINE OF BITS (CASPIPO)
    WITH STATES  ( m0,m1 );
m11: NODE;
m12: NODE;

oo: MACHINE OF BITS (LA2)
    WITH STATES  ( o0,o1 );
o11: NODE;
o12: NODE;

pp: MACHINE OF BITS (ACC_PEND)
    WITH STATES  ( p0,p1 );
p11: NODE;
p12: NODE;

qq: MACHINE OF BITS (RASE)
    WITH STATES  ( q0,q1 );
q11: NODE;
q12: NODE;

rr: MACHINE OF BITS (RASO)
    WITH STATES  ( r0,r1 );
r11: NODE;
```

```
r12: NODE;

tt: MACHINE OF BITS (LA3)
    WITH STATES  ( t0,t1 );
t11: NODE;
t12: NODE;

uu: MACHINE OF BITS (RDY)
    WITH STATES  ( u0,u1 );
u11: NODE;
u12: NODE;

ccc: MACHINE OF BITS (REFING)
    WITH STATES  ( ccc0,ccc1 );
cc11: NODE;
cc12: NODE;

vv: MACHINE OF BITS ( COUNT1[3..0] )
    WITH STATES  (
      Z10   = B"0000",
      Z11   = B"0001",
      Z12   = B"0010",
      Z13   = B"0011",
      Z14   = B"0100",
      Z15   = B"0101",
      Z16   = B"0110",
      Z17   = B"0111",
      Z18   = B"1000",
      Z19   = B"1001",
      Z110  = B"1010",
      Z111  = B"1011",
      Z112  = B"1100",
      Z113  = B"1101",
      Z114  = B"1110",
      Z115  = B"1111");
j111 : NODE;
k111 : NODE;

ww: MACHINE OF BITS ( COUNT2[3..0] )
    WITH STATES  (
      Z20   = B"0000",
      Z21   = B"0001",
      Z22   = B"0010",
      Z23   = B"0011",
      Z24   = B"0100",
      Z25   = B"0101",
      Z26   = B"0110",
      Z27   = B"0111",
```

```
      Z28   = B"1000",
      Z29   = B"1001",
      Z210  = B"1010",
      Z211  = B"1011",
      Z212  = B"1100",
      Z213  = B"1101",
      Z214  = B"1110",
      Z215  = B"1111");
w11  : NODE;
w12  : NODE;
w13  : NODE;
w14  : NODE;


xx: MACHINE OF BITS (REFEVEN)
    WITH STATES  ( x0,x1 );
x11  : NODE;

nn: MACHINE OF BITS ( QA[4..0] )
    WITH STATES  (
        IDLE = B"00000",
     ACCESS0 = B"00001",
     ACCESS1 = B"00010",
     ACCESS2 = B"00011",
     ACCESS3 = B"00100",
     ACCESS4 = B"00101",
     ACCESS5 = B"00110",
     ACCESS6 = B"00111",
     ACCESS7 = B"01000",
     ACCESS8 = B"01001",
    REFRESH0 = B"11010",
    REFRESH1 = B"11011",
    REFRESH2 = B"11100",
    REFRESH3 = B"11101",
    REFRESH4 = B"11110",
    REFRESH5 = B"11111");
na: NODE;
nb: NODE;
nc: NODE;
nd: NODE;
ne: NODE;
nf: NODE;
ng: NODE;
nh: NODE;
ni: NODE;
nj: NODE;
nk: NODE;
nl: NODE;
```

```
 nm: NODE;
 no: NODE;
 np: NODE;


 RDEN     : NODE;
 WAIT     : NODE;
 WRE      : NODE;
 WRO      : NODE;
 REFREQ   : NODE;
 DRAMADDR : NODE;
 HLDACK   : NODE;
/WRE      : NODE;
/WRO      : NODE;
 SRASE    : NODE;
 XTEND    : NODE;
 SXTEND   : NODE;
 RASE_0   : NODE;
 RASE_1   : NODE;
 RASO_0   : NODE;
 RASO_1   : NODE;


 BEGIN


/RDEN     = !RDEN;
/BANKSELA = !BANKSELA;
/BANKSELB = !BANKSELB;
/RDY      = TRI(!RDY, (!QA4 & RASE));
/ADDRMUX  = !ADDRMUX;
/CASEE_B0 = !CASEE_B0;
/CASEE_B1 = !CASEE_B1;
/CASEE_B2 = !CASEE_B2;
/CASEE_B3 = !CASEE_B3;
/CASOO_B0 = !CASOO_B0;
/CASOO_B1 = !CASOO_B1;
/CASOO_B2 = !CASOO_B2;
/CASOO_B3 = !CASOO_B3;
/WRE      = !WRE;
/WRO      = !WRO;
/RASE_0   = !RASE_0;
/RASE_1   = !RASE_1;
/RASO_0   = !RASO_0;
/RASO_1   = !RASO_1;
/RASE     = !RASE;
/RASO     = !RASO;
 SRASE    =  AAA;
```

```
 SXTEND   =  BBB;
 HLDACK   =  HOLDACK;
 XTEND    =  EXTEND # !HLDACK # W_/R;


% RAS0, RAS2 FOR EVEN BANK %
RASE_0 = ( !PD2 & !PD1 & !PD0 & RASE )
    # ( !PD2 & !PD1 & PD0 & !A21 & RASE )
    # ( !PD2 & PD1 & !PD0 & RASE )
    # ( !PD2 & PD1 & PD0 & !A23 & RASE )
    # ( REFING & RASE )
    # ( PD2 & !PD1 & !PD0 & RASE)
    # ( PD2 & !PD1 & PD0 & !A25 & RASE );


% RAS1, RAS3 FOR EVEN BANK %
RASE_1 = ( !PD2 & !PD1 & PD0 & A21 & RASE )
    # ( !PD2 & PD1 & PD0 & A23 & RASE )
    # ( PD2 & !PD1 & PD0 & A25 & RASE )
    # ( REFING & RASE );


% RAS0, RAS2 FOR ODD BANK %
RASO_0 = ( !PD2 & !PD1 & !PD0 & RASO )
    # ( !PD2 & !PD1 & PD0 & !A21 & RASO )
    # ( !PD2 & PD1 & !PD0 & RASO )
    # ( !PD2 & PD1 & PD0 & !A23 & RASO )
    # ( REFING & RASO )
    # ( PD2 & !PD1 & !PD0 & RASO )
    # ( PD2 & !PD1 & PD0 & !A25 & RASO );


% RAS1, RAS3 FOR ODD BANK %
RASO_1 = ( !PD2 & !PD1 & PD0 & A21 & RASO )
    # ( !PD2 & PD1 & PD0 & A23 & RASO )
    # ( PD2 & !PD1 & PD0 & A25 & RASO )
    # ( REFING & RASO );


% DRAMADDR IS BEING DECODED USING A[30:27] OF ADDR BUS %
DRAMADDR = ( !A30 & !A29 & !A28 & !A27 );


% DEASSERT CAS SIGNALS IF BLAST ACTIVE %
WAIT = ( IDLE & !W_/R )
       # ( ACCESS0 & !W_/R )
       # ( ACCESS1 & !W_/R )
       # ( ACCESS2 & !W_/R )
```

```
       # ( ACCESS3 & !W_/R & !LA2 );


% REFRESH WHEN BOTH COUNTERS COUNTDOWN TO 0 %
REFREQ = ( !COUNT13 & !COUNT12 & !COUNT11 & !COUNT10
     & !COUNT23 & !COUNT22 & !COUNT21 & !COUNT20 );


% DATA READ OUTPUT ENABLE %
RDEN = !QA4 & !W_/R & RASE;


% EVEN BANK DATA WRITE ENABLE %
WRE = !QA4 & W_/R & RASE;


% ODD BANK DATA WRITE ENABLE %
WRO = !QA4 & W_/R & RASE;


% ********************************************************** %
%        A3ODD                                               %
% ********************************************************** %

z = ( IDLE & A3 & !DCLK1 )
    # ( ACCESS4 & !W_/R & DCLK1 & LA3 & XTEND )
    # ( ACCESS5 & W_/R & !DCLK1 );
y = ( IDLE & !A3  & !DCLK1 )
    # ( ACCESS4 & !W_/R & DCLK1 & !LA3 & XTEND )
    # ( ACCESS5 & W_/R & !DCLK1 );
ss.CLK   = CLK2;
ss.RESET = !/RESET;

TABLE

% CURRENT STATE       INPUT1            INPUT2        NEXT STATE %

ss,                     y,                z          => ss;
s1,                     1,                x          => s0;
s1,                     0,                x          => s1;
s0,                     x,                1          => s1;
s0,                     x,                0          => s0;

END TABLE;
```

```
% ********************************************************** %
%        A3EVEN                                              %
% ********************************************************** %

b = ( IDLE & A3 & !A2 & !DCLK1 )
    # ( IDLE & !A3 & A2 & !DCLK1 )
    # ( ACCESS3 & LA3 & LA2 & DCLK1 & !W_/R & XTEND )
    # ( ACCESS5 & !LA3 & !LA2 & DCLK1 & !W_/R )
    # ( ACCESS3 & LA3 & LA2 & !DCLK1 & W_/R & HLDACK )
    # ( ACCESS7 & !LA3 & !LA2 & !DCLK1 & W_/R & HLDACK )
    # ( ACCESS4 & LA3 & LA2 & DCLK1 & W_/R & !HLDACK )
    # ( ACCESS6 & !LA3 & !LA2 & DCLK1 & W_/R & !HLDACK );
c = ( IDLE & !A3 & !A2 & !DCLK1 )
    # ( IDLE & A3 & A2 & !DCLK1 )
    # ( ACCESS3 & !LA3 & LA2 & DCLK1 & !W_/R & XTEND )
    # ( ACCESS5 & LA3 & !LA2 & DCLK1 & !W_/R )
    # ( ACCESS3 & !LA3 & LA2 & !DCLK1 & W_/R & HLDACK )
    # ( ACCESS7 & LA3 & !LA2 & !DCLK1 & W_/R & HLDACK )
    # ( ACCESS4 & !LA3 & LA2 & DCLK1 & W_/R & !HLDACK )
    # ( ACCESS6 & LA3 & !LA2 & DCLK1 & W_/R & !HLDACK );
yy.CLK   = CLK2;
yy.RESET = !/RESET;

TABLE

% CURRENT STATE       INPUT1            INPUT2        NEXT STATE %

yy,                     c,                b          => yy;
y1,                     1,                x          => y0;
y1,                     0,                x          => y1;
y0,                     x,                1          => y1;
y0,                     x,                0          => y0;

END TABLE;


% ********************************************************** %
%        ADDRMUX                                             %
% ********************************************************** %

j = ( !RASE & DCLK1 );
k = ( RASE & DCLK1 );
zz.CLK   = CLK2;
zz.RESET = !/RESET;

TABLE
```

```
% CURRENT STATE       INPUT1            INPUT2        NEXT STATE %

zz,                    j,                k           => zz;
z1,                    1,                x           => z0;
z1,                    0,                x           => z1;
z0,                    x,                1           => z1;
z0,                    x,                0           => z0;

END TABLE;


% ********************************************************** %
%        BANKSELA                                            %
% ********************************************************** %

b11 = ( IDLE & A2 ) # ( ACCESS3 & XTEND ) # ( ACCESS5 );
b12 = ( IDLE & !A2 ) # ( ACCESS4 & XTEND );
bb.CLK   = CLK1;
bb.RESET = !/RESET;

TABLE

% CURRENT STATE       INPUT1            INPUT2        NEXT STATE %

bb,                   b11,               b12         => bb;
b1,                    1,                x           => b0;
b1,                    0,                x           => b1;
b0,                    x,                1           => b1;
b0,                    x,                0           => b0;

END TABLE;


% ********************************************************** %
%        BANKSELB                                            %
% ********************************************************** %

c11 = ( IDLE & A2 ) # ( ACCESS3 & XTEND ) # ( ACCESS5 );
c12 = ( IDLE & !A2 ) # ( ACCESS4 & XTEND );
cc.CLK   = CLK1;
cc.RESET = !/RESET;

TABLE

% CURRENT STATE       INPUT1            INPUT2        NEXT STATE %

cc,                   c11,               c12         => cc;
c1,                    1,                x           => c0;
```

```
c1,                    0,                 x          => c1;
c0,                    x,                 1          => c1;
c0,                    x,                 0          => c0;

END TABLE;


% ************************************************************ %
%        CAS EVEN BYTE ENABLE 0                                %
% ************************************************************ %

d11 = ( !QA4 & !CASPIPE )
      # ( !QA4 & !WAIT & !/BLAST & !W_/R & !DCLK1 & XTEND )
      # ( QA4 & !CASPIPE );
d12 = ( !QA4 & CASPIPE & !/BE0 )
      # ( QA4 & CASPIPE );
dd.CLK   = CLK2;
dd.RESET = !/RESET;

TABLE

% CURRENT STATE       INPUT1            INPUT2        NEXT STATE %

dd,                    d11,               d12          => dd;
d1,                    1,                 x            => d0;
d1,                    0,                 x            => d1;
d0,                    x,                 1            => d1;
d0,                    x,                 0            => d0;

END TABLE;

% ************************************************************ %
%        CAS EVEN BYTE ENABLE 1                                %
% ************************************************************ %

e11 = ( !QA4 & !CASPIPE )
      # ( !QA4 & !WAIT & !/BLAST & !W_/R & !DCLK1 & XTEND )
      # ( QA4 & !CASPIPE );
e12 = ( !QA4 & CASPIPE & !/BE1 )
      # ( QA4 & CASPIPE );
ee.CLK   = CLK2;
ee.RESET = !/RESET;

TABLE

% CURRENT STATE       INPUT1            INPUT2        NEXT STATE %

ee,                    e11,               e12          => ee;
```

```
e1,                     1,                  x           => e0;
e1,                     0,                  x           => e1;
e0,                     x,                  1           => e1;
e0,                     x,                  0           => e0;

END TABLE;


% ********************************************************** %
%        CAS EVEN BYTE ENABLE 2                              %
% ********************************************************** %

f11 = ( !QA4 & !CASPIPE )
      # ( !QA4 & !WAIT & !/BLAST & !W_/R & !DCLK1 & XTEND )
      # ( QA4 & !CASPIPE );
f12 = ( !QA4 & CASPIPE & !/BE2 )
      # ( QA4 & CASPIPE );
ff.CLK   = CLK2;
ff.RESET = !/RESET;

TABLE

% CURRENT STATE       INPUT1            INPUT2        NEXT STATE %

ff,                    f11,               f12          => ff;
f1,                     1,                 x           => f0;
f1,                     0,                 x           => f1;
f0,                     x,                 1           => f1;
f0,                     x,                 0           => f0;

END TABLE;


% ********************************************************** %
%        CAS EVEN BYTE ENABLE 3                              %
% ********************************************************** %

g11 = ( !QA4 & !CASPIPE )
      # ( !QA4 & !WAIT & !/BLAST & !W_/R & !DCLK1 & XTEND )
      # ( QA4 & !CASPIPE );
g12 = ( !QA4 & CASPIPE & !/BE3 )
      # ( QA4 & CASPIPE );
gg.CLK   = CLK2;
gg.RESET = !/RESET;

TABLE
```

```
% CURRENT STATE       INPUT1            INPUT2        NEXT STATE %

gg,                    g11,               g12          => gg;
g1,                     1,                x            => g0;
g1,                     0,                x            => g1;
g0,                     x,                1            => g1;
g0,                     x,                0            => g0;

END TABLE;


% ********************************************************** %
%        CAS ODD BYTE ENABLE 0                               %
% ********************************************************** %

h11 = ( !QA4 & !CASPIPO )
      # ( !QA4 & !WAIT & !/BLAST & !W_/R & !DCLK1 & XTEND )
      # ( QA4 & !CASPIPO);
h12 = ( !QA4 & CASPIPO & !/BE0 )
      # ( QA4 & CASPIPO );
hh.CLK   = CLK2;
hh.RESET = !/RESET;

TABLE

% CURRENT STATE       INPUT1            INPUT2        NEXT STATE %

hh,                    h11,               h12          => hh;
h1,                     1,                x            => h0;
h1,                     0,                x            => h1;
h0,                     x,                1            => h1;
h0,                     x,                0            => h0;

END TABLE;


% ********************************************************** %
%        CAS ODD BYTE ENABLE 1                               %
% ********************************************************** %

i11 = ( !QA4 & !CASPIPO )
      # ( !QA4 & !WAIT & !/BLAST & !W_/R & !DCLK1 & XTEND )
      # ( QA4 & !CASPIPO );
i12 = ( !QA4 & CASPIPO & !/BE1 )
      # ( QA4 & CASPIPO );
ii.CLK   = CLK2;
ii.RESET = !/RESET;
```

```
TABLE

% CURRENT STATE       INPUT1            INPUT2        NEXT STATE %

ii,                    i11,               i12          => ii;
i1,                     1,                 x           => i0;
i1,                     0,                 x           => i1;
i0,                     x,                 1           => i1;
i0,                     x,                 0           => i0;

END TABLE;


% ********************************************************** %
%        CAS ODD BYTE ENABLE 2                               %
% ********************************************************** %

j11 = ( !QA4 & !CASPIPO )
      # ( !QA4 & !WAIT & !/BLAST & !W_/R & !DCLK1 & XTEND )
      # ( QA4 & !CASPIPO );
j12 = ( !QA4 & CASPIPO & !/BE2 )
      # ( QA4 & CASPIPO );
jj.CLK   = CLK2;
jj.RESET = !/RESET;

TABLE

% CURRENT STATE       INPUT1            INPUT2        NEXT STATE %

jj,                    j11,               j12          => jj;
j1,                     1,                 x           => j0;
j1,                     0,                 x           => j1;
j0,                     x,                 1           => j1;
j0,                     x,                 0           => j0;

END TABLE;


% ********************************************************** %
%        CAS ODD BYTE ENABLE 3                               %
% ********************************************************** %

k11 = ( !QA4 & !CASPIPO )
      # ( !QA4 & !WAIT & !/BLAST & !W_/R & !DCLK1 & XTEND )
      # ( QA4 & !CASPIPO );
k12 = ( !QA4 & CASPIPO & !/BE3 )
      # ( QA4 & CASPIPO );
kk.CLK   = CLK2;
```

```
kk.RESET = !/RESET;

TABLE

% CURRENT STATE       INPUT1            INPUT2        NEXT STATE %

kk,                    k11,              k12           => kk;
k1,                     1,                x            => k0;
k1,                     0,                x            => k1;
k0,                     x,                1            => k1;
k0,                     x,                0            => k0;

END TABLE;


% ********************************************************** %
%        CASPIPE                                             %
% ********************************************************** %

l11 = ( ACCESS3 & DCLK1 & !W_/R & XTEND )
      # ( XTEND & SXTEND & !DCLK1 & !/BLAST )
      # ( ACCESS5 & DCLK1 & !W_/R )
      # ( ACCESS4 & !DCLK1 & !/BLAST & !W_/R )
      # ( ACCESS3 & DCLK1 & HLDACK & W_/R )
      # ( ACCESS5 & DCLK1 & W_/R & HLDACK )
      # ( ACCESS7 & DCLK1 & W_/R & HLDACK )
      # ( REFRESH2 & DCLK1 )
      # ( ACCESS3 & !DCLK1 & W_/R & !HLDACK )
      # ( ACCESS5 & !DCLK1 & W_/R & !HLDACK )
      # ( ACCESS7 & !DCLK1 & W_/R & !HLDACK );
l12 = ( ACCESS1 & !W_/R & !DCLK1 )
      # ( ACCESS3 & !W_/R & !DCLK1 & /BLAST & XTEND )
      # ( ACCESS5 & !W_/R & !DCLK1 & /BLAST & !LA2 )
      # ( ACCESS2 & W_/R & DCLK1 & HLDACK )
      # ( ACCESS6 & W_/R & DCLK1 & HLDACK )
      # ( REFRESH0 & DCLK1 & REFEVEN )
      # ( ACCESS2 & W_/R & !DCLK1 & !HLDACK & LA2 )
      # ( ACCESS4 & W_/R & !DCLK1 & !HLDACK & /BLAST )
      # ( ACCESS6 & W_/R & !DCLK1 & !HLDACK & /BLAST );
ll.CLK   = CLK2;
ll.RESET = !/RESET;

TABLE

% CURRENT STATE       INPUT1            INPUT2        NEXT STATE %

ll,                    l11,              l12           => ll;
l1,                     1,                x            => l0;
```

```
l1,                     0,                  x           => l1;
l0,                     x,                  1           => l1;
l0,                     x,                  0           => l0;

END TABLE;


% ********************************************************** %
%        CASPIPO                                             %
% ********************************************************** %

m11 = ( ACCESS4 & !W_/R & DCLK1 & XTEND )
      # ( XTEND & SXTEND & !DCLK1 & !/BLAST )
      # ( ACCESS6 & !W_/R  & DCLK1  )
      # ( ACCESS5 & !DCLK1 & !/BLAST & !W_/R )
      # ( ACCESS5 & W_/R & DCLK1 & HLDACK)
      # ( REFRESH2 & DCLK1 )
      # ( ACCESS4 & W_/R & !DCLK1 & !HLDACK )
      # ( ACCESS6 & W_/R & !DCLK1 & !HLDACK );
m12 = ( ACCESS2 & !W_/R & LA2 & /BLAST & !DCLK1 & XTEND )
      # ( ACCESS2 & !W_/R & !LA2  & !DCLK1 )
      # ( ACCESS4 & !W_/R & !DCLK1 & /BLAST & XTEND )
      # ( ACCESS4 & W_/R & DCLK1 & HLDACK )
      # ( REFRESH0 & DCLK1 & !REFEVEN )
      # ( ACCESS5 & W_/R & !DCLK1 & /BLAST & !HLDACK )
      # ( ACCESS3 & W_/R & !DCLK1 & /BLAST & !HLDACK )
      # ( ACCESS3 & W_/R & !DCLK1 & !LA2 & !HLDACK );
mm.CLK   = CLK2;
mm.RESET = !/RESET;

TABLE

% CURRENT STATE       INPUT1            INPUT2        NEXT STATE %

mm,                    m11,               m12          => mm;
m1,                     1,                 x           => m0;
m1,                     0,                 x           => m1;
m0,                     x,                 1           => m1;
m0,                     x,                 0           => m0;

END TABLE;


% ********************************************************** %
%        REFEVEN                                             %
% ********************************************************** %

x11 = ( REFRESH1 & !DCLK1 );
```

```
xx.CLK   = CLK2;
xx.RESET = !/RESET;

TABLE

% CURRENT STATE       INPUT1           NEXT STATE %

xx,                    x11               => xx;
x1,                     1                => x0;
x1,                     0                => x1;
x0,                     1                => x1;
x0,                     0                => x0;

END TABLE;


% ********************************************************** %
%         LA2                                                %
% ********************************************************** %

o11 = ( IDLE & A2 );
o12 = ( IDLE & !A2 );
oo.CLK   = CLK1;
oo.RESET = !/RESET;

TABLE

% CURRENT STATE       INPUT1            INPUT2        NEXT STATE %

oo,                    o11,               o12          => oo;
o1,                     1,                x            => o0;
o1,                     0,                x            => o1;
o0,                     x,                1            => o1;
o0,                     x,                0            => o0;

END TABLE;


% ********************************************************** %
%        LA3                                                 %
% ********************************************************** %

t11 = ( IDLE & A3 );
t12 = ( IDLE & !A3 );
tt.CLK   = CLK1;
tt.RESET = !/RESET;

TABLE
```

```
% CURRENT STATE       INPUT1            INPUT2        NEXT STATE %

tt,                   t11,                t12          => tt;
t1,                    1,                 x            => t0;
t1,                    0,                 x            => t1;
t0,                    x,                 1            => t1;
t0,                    x,                 0            => t0;

END TABLE;


% ********************************************************** %
%        REFING                                              %
% ********************************************************** %

cc11 = ( IDLE );
cc12 = ( REFRESH0 );
ccc.CLK   = CLK1;
ccc.RESET = !/RESET;

TABLE

% CURRENT STATE       INPUT1            INPUT2        NEXT STATE %

ccc,                  cc11,              cc12         => ccc;
ccc1,                  1,                 x           => ccc0;
ccc1,                  0,                 x           => ccc1;
ccc0,                  x,                 1           => ccc1;
ccc0,                  x,                 0           => ccc0;

END TABLE;


% ********************************************************** %
%        ACC_PEND                                            %
% ********************************************************** %

p11 = ACCESS3;
p12 = !/ADS & DRAMADDR;
pp.CLK   = CLK1;
pp.RESET = !/RESET;

TABLE

% CURRENT STATE       INPUT1            INPUT2        NEXT STATE %

pp,                    p11,                p12          => pp;
```

```
p1,                     1,                  x           => p0;
p1,                     0,                  x           => p1;
p0,                     x,                  1           => p1;
p0,                     x,                  0           => p0;

END TABLE;


% ********************************************************** %
%        RASE                                                %
% ********************************************************** %

q11 = ( ACCESS3 & !DCLK1 & LA2 & !/BLAST & XTEND )
      # ( ACCESS4 & !DCLK1 & !W_/R & !/BLAST & XTEND )
      # ( ACCESS4 & !DCLK1 & W_/R & !/BLAST & !HLDACK )
      # ( ACCESS5 & !DCLK1 & !/BLAST )
      # ( ACCESS7 & !DCLK1 & !/BLAST )
      # ( ACCESS6 & !DCLK1 & !/BLAST & !W_/R )
      # ( ACCESS6 & !DCLK1 & !/BLAST & W_/R & !HLDACK )
      # ( REFRESH3 & !DCLK1 );
q12 = ( IDLE & !SRASE & !/ADS & !REFREQ & !ACC_PEND & DRAMADDR & !DCLK1
                        )
      # ( ACCESS0 & !DCLK1 )
      # ( REFRESH1 & DCLK1 & REFEVEN );
qq.CLK   = CLK2;
qq.RESET = !/RESET;

TABLE

% CURRENT STATE       INPUT1            INPUT2        NEXT STATE %

qq,                   q11,               q12           => qq;
q1,                     1,                 x           => q0;
q1,                     0,                 x           => q1;
q0,                     x,                 1           => q1;
q0,                     x,                 0           => q0;

END TABLE;


% ********************************************************** %
%        RASO                                                %
% ********************************************************** %

r11 = ( ACCESS3 & !DCLK1 & LA2 & !/BLAST & XTEND )
      # ( ACCESS4 & !DCLK1 & !W_/R & !/BLAST & XTEND )
      # ( ACCESS4 & !DCLK1 & W_/R & !/BLAST & !HLDACK )
      # ( ACCESS5 & !DCLK1 & !/BLAST )
```

```
      # ( ACCESS7 & !DCLK1 & !/BLAST )
      # ( ACCESS6 & !DCLK1 & !/BLAST & !W_/R )
      # ( ACCESS6 & !DCLK1 & !/BLAST & W_/R & !HLDACK )
      # ( REFRESH3 & !DCLK1 );
r12 = ( IDLE & !SRASE & !/ADS & !REFREQ & !ACC_PEND & DRAMADDR & !DCLK1
                          )
      # ( ACCESS0 & !DCLK1 )
      # ( REFRESH1 & DCLK1 & !REFEVEN );
rr.CLK   = CLK2;
rr.reset = !/RESET;

TABLE

% CURRENT STATE       INPUT1            INPUT2        NEXT STATE %

rr,                    r11,              r12          => rr;
r1,                     1,               x            => r0;
r1,                     0,               x            => r1;
r0,                     x,               1            => r1;
r0,                     x,               0            => r0;

END TABLE;


% ********************************************************** %
%        RDY                                                 %
% ********************************************************** %

u11 = ( !W_/R & !/BLAST & XTEND )
      # ( W_/R & /blast & HLDACK )
      # ( W_/R & !/blast );
u12 = ( ACCESS2 & LA2  )
      # ( ACCESS3 & !LA2 & !HLDACK & W_/R )
      # ( ACCESS3 & !W_/R & !LA2 )
      # ( ACCESS4 & W_/R & HLDACK )
      # ( ACCESS6 & HLDACK & W_/R );
uu.CLK   = CLK1;
uu.RESET = !/RESET;

TABLE

% CURRENT STATE       INPUT1            INPUT2        NEXT STATE %

uu,                    u11,              u12          => uu;
u1,                     1,               x            => u0;
u1,                     0,               x            => u1;
u0,                     x,               1            => u1;
u0,                     x,               0            => u0;
```

```
END TABLE;

% ********************************************************** %
%        SRASE                                               %
% ********************************************************** %

AAA = RASE # RASO;
AAA.CLK = CLK1;


% ********************************************************** %
%        SXTEND                                              %
% ********************************************************** %

BBB = !XTEND;
BBB.CLK = CLK1;


% ********************************************************** %
%        MAIN STATE MACHINE                                  %
% ********************************************************** %

na = ( !/ADS & !ACC_PEND & DRAMADDR & SRASE & !REFREQ );
nb = ( !/ADS & !ACC_PEND & DRAMADDR & !SRASE & !REFREQ & !A2 );
nc = ( !/ADS & !ACC_PEND & DRAMADDR & !SRASE & !REFREQ & A2 & !W_/R );
nd = ( !/ADS & !ACC_PEND & DRAMADDR & !SRASE & !REFREQ & A2 & W_/R );
ne = (  ACC_PEND & !REFREQ );
nf = (  REFREQ );
ng = ( !W_/R & !LA2 );
nh = (  W_/R & !LA2 );
ni = ( !/BLAST & LA2 );
nj = ( !/BLAST & !W_/R );
nk = ( !/BLAST );
nl = (  W_/R & HLDACK );
nm = (  W_/R & !HLDACK & !/BLAST );
no = ( !XTEND );
np = ( W_/R & !HLDACK );

nn.CLK   = CLK1;
nn.RESET = !/RESET;

TABLE
```

```
% CURRENT STATE      CURRENT INPUTs
     NEXT STATE       %

nn,                na, nb, nc, nd, ne, nf, ng, nh, ni, nj, nk, nl, nm,
                       no, np
     =>  nn;
IDLE,               1,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,
                       x,  x
     => ACCESS0;
IDLE,               0,  1,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,
                       x,  x
     => ACCESS1;
IDLE,               0,  0,  1,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,
                       x,  x
     => ACCESS2;
IDLE,               0,  0,  0,  1,  x,  x,  x,  x,  x,  x,  x,  x,  x,
                       x,  0
     => ACCESS3;
IDLE,               0,  0,  0,  1,  x,  x,  x,  x,  x,  x,  x,  x,  x,
                       x,  1
     => ACCESS2;
IDLE,               0,  0,  0,  0,  1,  x,  x,  x,  x,  x,  x,  x,  x,
                       x,  x
     => ACCESS0;
IDLE,               0,  0,  0,  0,  0,  1,  x,  x,  x,  x,  x,  x,  x,
                       x,  x
     => REFRESH0;
IDLE,               0,  0,  0,  0,  0,  0,  x,  x,  x,  x,  x,  x,  x,
                       x,  x
     => IDLE;
```

```
% nn                na, nb, nc, nd, ne, nf, ng, nh, ni, nj, nk, nl, nm,
                          no, np %

ACCESS0,            x,  x,  x,  x,  x,  x,  1,  x,  x,  x,  x,  x,  x,
                          x,  x
     => ACCESS2;
ACCESS0,            x,  x,  x,  x,  x,  x,  0,  1,  x,  x,  x,  x,  x,
                          x,  0
     => ACCESS3;
ACCESS0,            x,  x,  x,  x,  x,  x,  0,  1,  x,  x,  x,  x,  x,
                          x,  1
     => ACCESS2;
ACCESS0,            x,  x,  x,  x,  x,  x,  0,  0,  x,  x,  x,  x,  x,
                          x,  x
     => ACCESS1;

ACCESS1,            x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,
                          x,  x
     => ACCESS2;

ACCESS2,            x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,
                          x,  x
     => ACCESS3;

ACCESS3,            x,  x,  x,  x,  x,  x,  x,  x,  1,  x,  x,  x,  x,
                          0,  x
     => IDLE;
ACCESS3,            x,  x,  x,  x,  x,  x,  x,  x,  0,  x,  x,  x,  x,
                          0,  x
     => ACCESS4;
ACCESS3,            x,  x,  x,  x,  x,  x,  0,  x,  x,  x,  x,  x,  x,
                          1,  x
     => ACCESS3;
ACCESS3,            x,  x,  x,  x,  x,  x,  1,  x,  x,  x,  x,  x,  x,
                          1,  x
     => ACCESS4;
```

```
% nn                na, nb, nc, nd, ne, nf, ng, nh, ni, nj, nk, nl, nm,
                       no, np %

ACCESS4,            x,  x,  x,  x,  x,  x,  x,  x,  x,  1,  x,  x,  x,
                       0,  x
     => IDLE;
ACCESS4,            x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  1,
                       0,  x
     => IDLE;
ACCESS4,            x,  x,  x,  x,  x,  x,  x,  x,  x,  0,  x,  x,  0,
                       0,  x
     => ACCESS5;
ACCESS4,            x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,
                       1,  x
     => ACCESS4;

ACCESS5,            x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  1,  x,  x,
                       x,  x
     => IDLE;
ACCESS5,            x,  x,  x,  x,  x,  x,  1,  x,  x,  x,  0,  x,  x,
                       x,  x
     => ACCESS4;
ACCESS5,            x,  x,  x,  x,  x,  x,  0,  x,  x,  x,  0,  x,  x,
                       x,  x
     => ACCESS6;
```

```
% nn                na, nb, nc, nd, ne, nf, ng, nh, ni, nj, nk, nl, nm,
                       no, np %

ACCESS6,            x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  1,  0,  x,
                       x,  x
     => IDLE;
ACCESS6,            x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  1,  x,
                       x,  x
     => ACCESS7;
ACCESS6,            x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  0,  x,  x,
                       x,  1
     => ACCESS7;

ACCESS7,            x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  1,  x,  x,
                       x,  x
     => IDLE;
ACCESS7,            x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  0,  x,  x,
                       x,  x
     => ACCESS4;

ACCESS8,            x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,
                       x,  x
     => IDLE;
```

```
% nn                na, nb, nc, nd, ne, nf, ng, nh, ni, nj, nk, nl, nm,
                       no, np %

REFRESH0,           x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,
                       x,  x
     => REFRESH1;
REFRESH1,           x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,
                       x,  x
     => REFRESH2;
REFRESH2,           x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,
                       x,  x
     => REFRESH3;
REFRESH3,           x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,
                       x,  x
     => IDLE;
REFRESH4,           x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,
                       x,  x
     => IDLE;
REFRESH5,           x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,  x,
                       x,  x
     => IDLE;

END TABLE;


% ********************************************************** %
%        REFRESH LOWER COUNTER                               %
% ********************************************************** %

j111 = ( !QA4 & Z20 );
k111 = QA4;
vv.CLK   = CLK1;
vv.RESET = !/RESET;

TABLE

% CURRENT STATE      CURRENT INPUTs          NEXT STATE %

vv,                j111, k111              => vv;
Z10,                 1,   x                => Z10;
Z10,                 0,   x                => Z115;
Z11,                 x,   1                => Z115;
Z11,                 x,   0                => Z10;
Z12,                 x,   1                => Z115;
Z12,                 x,   0                => Z11;
Z13,                 x,   1                => Z115;
Z13,                 x,   0                => Z12;
Z14,                 x,   1                => Z115;
```

```
Z14,                x,   0                => Z13;
Z15,                x,   1                => Z115;
Z15,                x,   0                => Z14;
Z16,                x,   1                => Z115;
Z16,                x,   0                => Z15;
Z17,                x,   1                => Z115;
Z17,                x,   0                => Z16;
Z18,                x,   1                => Z115;
Z18,                x,   0                => Z17;
Z19,                x,   1                => Z115;
Z19,                x,   0                => Z18;
Z110,               x,   1                => Z115;
Z110,               x,   0                => Z19;
Z111,               x,   1                => Z115;
Z111,               x,   0                => Z110;
Z112,               x,   1                => Z115;
Z112,               x,   0                => Z111;
Z113,               x,   1                => Z115;
Z113,               x,   0                => Z112;
Z114,               x,   1                => Z115;
Z114,               x,   0                => Z113;
Z115,               x,   1                => Z115;
Z115,               x,   0                => Z114;

END TABLE;


% ********************************************************** %
%        REFRESH UPPER COUNTER                               %
% ********************************************************** %

w11 = Z10;
w12 = QA4;
w13 = !25/33sw;
w14 = s5mhz;
ww.CLK   = CLK1;
ww.RESET = !/RESET;

TABLE

% CURRENT STATE      CURRENT INPUTs               NEXT STATE   %

ww,                w11, w12, W13, w14             => ww;

Z20,                x,   1,   0,   0              => Z215;
Z20,                x,   0,   x,   x              => Z20;
Z20,                x,   1,   1,   0              => Z211;
Z20,                x,   1,   x,   1              => Z21;
```

```
Z21,        x,   1,   0,   0        => Z215;
Z21,        1,   0,   x,   x        => Z20;
Z21,        0,   0,   x,   x        => Z21;
Z21,        x,   1,   1,   0        => Z211;
Z21,        x,   1,   x,   1        => Z21;

Z22,        x,   1,   0,   0        => Z215;
Z22,        1,   0,   x,   x        => Z21;
Z22,        0,   0,   x,   x        => Z22;
Z22,        x,   1,   1,   0        => Z211;
Z22,        x,   1,   x,   1        => Z21;

Z23,        x,   1,   0,   0        => Z215;
Z23,        1,   0,   x,   x        => Z22;
Z23,        0,   0,   x,   x        => Z23;
Z23,        x,   1,   1,   0        => Z211;
Z23,        x,   1,   x,   1        => Z21;

Z24,        x,   1,   0,   0        => Z215;
Z24,        1,   0,   x,   x        => Z23;
Z24,        0,   0,   x,   x        => Z24;
Z24,        x,   1,   1,   0        => Z211;
Z24,        x,   1,   x,   1        => Z21;

Z25,        x,   1,   0,   0        => Z215;
Z25,        1,   0,   x,   x        => Z24;
Z25,        0,   0,   x,   x        => Z25;
Z25,        x,   1,   1,   0        => Z211;
Z25,        x,   1,   x,   1        => Z21;

Z26,        x,   1,   0,   0        => Z215;
Z26,        1,   0,   x,   x        => Z25;
Z26,        0,   0,   x,   x        => Z26;
Z26,        x,   1,   1,   0        => Z211;
Z26,        x,   1,   x,   1        => Z21;

Z27,        x,   1,   0,   0        => Z215;
Z27,        1,   0,   x,   x        => Z26;
Z27,        0,   0,   x,   x        => Z27;
Z27,        x,   1,   1,   0        => Z211;
Z27,        x,   1,   x,   1        => Z21;

Z28,        x,   1,   0,   0        => Z215;
Z28,        1,   0,   x,   x        => Z27;
Z28,        0,   0,   x,   x        => Z28;
Z28,        x,   1,   1,   0        => Z211;
Z28,        x,   1,   x,   1        => Z21;
```

```
Z29,                x,   1,   0,   0              => Z215;
Z29,                1,   0,   x,   x              => Z28;
Z29,                0,   0,   x,   x              => Z29;
Z29,                x,   1,   1,   0              => Z211;
Z29,                x,   1,   x,   1              => Z21;

Z210,               x,   1,   0,   0              => Z215;
Z210,               1,   0,   x,   x              => Z29;
Z210,               0,   0,   x,   x              => Z210;
Z210,               x,   1,   1,   0              => Z211;
Z210,               x,   1,   x,   1              => Z21;

Z211,               x,   1,   0,   0              => Z215;
Z211,               1,   0,   x,   x              => Z210;
Z211,               0,   0,   x,   x              => Z211;
Z211,               x,   1,   1,   0              => Z211;
Z211,               x,   1,   x,   1              => Z21;

Z212,               x,   1,   0,   0              => Z215;
Z212,               1,   0,   x,   x              => Z211;
Z212,               0,   0,   x,   x              => Z212;
Z212,               x,   1,   1,   0              => Z211;
Z212,               x,   1,   x,   1              => Z21;

Z213,               x,   1,   0,   0              => Z215;
Z213,               1,   0,   x,   x              => Z212;
Z213,               0,   0,   x,   x              => Z213;
Z213,               x,   1,   1,   0              => Z211;
Z213,               x,   1,   x,   1              => Z21;

Z214,               x,   1,   0,   0              => Z215;
Z214,               1,   0,   x,   x              => Z213;
Z214,               0,   0,   x,   x              => Z214;
Z214,               x,   1,   1,   0              => Z211;
Z214,               x,   1,   x,   1              => Z21;

Z215,               x,   1,   0,   0              => Z215;
Z215,               1,   0,   x,   x              => Z214;
Z215,               0,   0,   x,   x              => Z215;
Z215,               x,   1,   1,   0              => Z211;
Z215,               x,   1,   x,   1              => Z21;

END TABLE;


END;
```

C.3 SRAM Controller and Bus Arbiter

```
%************************************************************%
%********* SRAM CONTROLLER AND ARBITRATION LOGIC ************%
%************************************************************%


SUBDESIGN  SAGE

(

%************************************************************%
%******************  PRIMARY INPUT SIGNALS ******************%
%************************************************************%

CLK1           : INPUT ;  % 1X CLOCK %
DCLK1          : INPUT ;  % 1X CLOCK %
CLK2           : INPUT ;  % 2X CLOCK %
/RESET         : INPUT ;  % EXTERNAL RESET %
/ADS           : BIDIR ;  % ADDRESS STROBE %
/ALE           : INPUT ;  % 401GF Address Latch Enable Signal %
/BLAST         : INPUT ;  % BURST LAST %
W_/R           : INPUT ;  % PROCESSOR READ/WRITE %
A30            : INPUT ;  % ADDRESS A30 %
A29            : INPUT ;  % ADDRESS A29 %
A28            : INPUT ;  % ADDRESS A28 %
A27            : INPUT ;  % ADDRESS A27 %
A26            : INPUT ;  % ADDRESS A26 %
A25            : INPUT ;  % ADDRESS A25 %
A24            : INPUT ;  % ADDRESS A24 %
A23            : INPUT ;  % ADDRESS A23 %
A22            : INPUT ;  % ADDRESS A22 %
A21            : INPUT ;  % ADDRESS A21 %
A20            : INPUT ;  % ADDRESS A20 %
LA3            : INPUT ;  % ADDRESS A3  %
LA2            : INPUT ;  % ADDRESS A2  %
/BE3           : INPUT ;  % BYTE ENABLE 3 %
/BE2           : INPUT ;  % BYTE ENABLE 2 %
/BE1           : INPUT ;  % BYTE ENABLE 1/ADDRESS A1 %
/BE0           : INPUT ;  % BYTE ENABLE 0/ADDRESS A0 %
/PUSH_CRIT_INT : INPUT ;  % Critical Interrupt from DS1233 %
/PCI_SERR      : INPUT ;  % PCI System error signal %
INT_REMOVE     : INPUT ;  % Input to remove critical interrupt %
S_SPARE1       : INPUT ;  % SPARE INPUT %
S_SPARE2       : INPUT ;  % SPARE INPUT %
S_SPARE3       : INPUT ;  % SPARE INPUT %
```

```
S_SPARE4       : INPUT ;  % SPARE INPUT %
/ACK           : INPUT ;  % Ethernet register sync. hand shaking Signal
                          %
PRQ            : INPUT ;  % Ethernet Remote DMA hand shaking Signal %
/PRD           : INPUT ;  % Port Read control from ethernet controller
                          %
/OE            : INPUT ;  % Output Enable for peripherals %
/WE            : INPUT ;  % Write Enable for peripherals %
/SerE2_CS      : INPUT ;  % Serial EEPROM chip select from I/O
                          controller %
/ENET_CS       : INPUT ;  % Ethernet chip select from IO controller %
/LATCH_CS      : INPUT ;  % ETHERNET PRQ status and 646 latch chip
                          select %

%*************************************************%
%*************   OUTPUT SIGNALS    ***************%
% *************************************************%

/SRAM_OE       : OUTPUT ;  % SRAM OUTPUT ENABLE CONTROL %
/SRAM_WE0      : OUTPUT ;  % SRAM FIRST BYTE WRITE CONTROL %
/SRAM_WE1      : OUTPUT ;  % SRAM SECOND BYTE WRITE CONTROL %
/SRAM_WE2      : OUTPUT ;  % SRAM THIRD BYTE WRITE CONTROL %
/SRAM_WE3      : OUTPUT ;  % SRAM FOURTH BYTE WRITE CONTROL %
/401_RDY       : BIDIR  ;  % READY TO 401GF %
/401_CRIT_INT  : OUTPUT ;  % Critical Interrupt to the Processor 401GF
                          %
 D_SLA2        : OUTPUT ;
 D_SLA3        : OUTPUT ;

 % Output Pins for Ethernet ack. and DMA Latch control %
/WACK          : OUTPUT ;  % Ethernet Write acknowledge %
/RACK          : OUTPUT ;  % Ethernet Read acknowledge %
R_DMA_WR       : OUTPUT ;  % System Remote DMA write %
/646_OE        : OUTPUT ;  % Remote DMA Latch F646 output control
                          signal %
646_DIR        : OUTPUT ;  % Remote DMA Latch F646 data path direction
                          control signal %
SAB            : OUTPUT ;  % F646 ehternet latch Control signal to
                          Write Enet. registers %
SBA           : OUTPUT ;  % F646 ehternet latch Control signal to Read
                          Enet. registers %


 % Serial EEPROM Read/Write port Control Signals %

 SerE2_OE     : OUTPUT ;  % Serial EEPROM Read Port Control %
 /SerE2_WE    : OUTPUT ;  % Serial EEPROM Write Port Control %
```

```
 %
   As a host Adapter PLX 9060 has the highest priority,
   SQUALL has the second highest and next is 401GF.

   As a stand alone card, arbitration is  for 401GF, SQUALL module
   and Two PCI agents. The 401GF has the highest priority in
   all the agents. SQUALL has the second highest priority
   and then PCISLOT1 and PCISLOT2 respectively.

   NOTE : When SQUALL has the bus the 401GF can not take the
   bus from SQUALL. Because Squall doesn't have any signal to
   inform that the other agent needs bus.
 %

 % 401GF ARBITRATION SIGNALS %

 401_HOLD : OUTPUT;
 BUSREQ, 401_HOLDA : INPUT;

 % PLX 9060ES(PCI - 401GF) LOCAL CPU SIDE ARBITRATION SIGNALS %
 9060_HOLDA : OUTPUT;
 9060_HOLD  : INPUT;

 % SQUALL ARBITRATION SIGNALS %
 /SQBG : OUTPUT;
 /SQBR : INPUT;

 % PLX 9060ES PCI SIDE ARBITRATION SIGNALS %
 /9060_GNT_T : OUTPUT;
 /9060_REQ : INPUT;

 % ARBITRATION SIGNALS FOR PCI SLOTS %
 /PCI_REQ1, /PCI_REQ2 : INPUT;
 /PCI_GNT1, /PCI_GNT2 : OUTPUT;

 % OTHER SIGNALS REQUIRED %
 /FRAME, /IRDY   : INPUT; % PCI bus control signals %


% ARBSEL = 0 ----> FOR HOST ADAPTER ARBITRATION
  ARBSEL = 1 ----> FOR STAND ALONE BOARD ARBITRATION  %
  ARBSEL : INPUT;


)

VARIABLE
```

```
/9060_GNT       : NODE ;  % Input signal of 9060 bus grant signal tri-
                           state buffer %
401_X1          : DFF  ;  % Input signal of 401GF adress strobe signal
                           tri-state buffer %
/PUSH_CRIT_NODE : DFF  ;  % Temp. Signal used in synchronisation of
                           push button interrupt
                                to System clock (33/25Mhz) %
INT_REMOVE_NODE : DFF  ;
CLK21           : NODE ;  % Inverted 2x clock for Write state machine %
/D_WE           : DFF ; % Delayed Write Enable %
SLA2            : DFF ;
SLA3            : DFF ;
Temp            : DFF ; % Dummy flip flop used to add inverter delay
                          for SRAM Write enable clock %


    STSR : MACHINE OF BITS(SR)
           WITH STATES ( SR0 = B"0", SR1 = B"1");
   STRDY : MACHINE OF BITS(/S_READY)
           WITH STATES ( S_READY0 = B"1", S_READY1 = B"0");
   STSOE : MACHINE OF BITS(/SRAM_OE)
          WITH STATES ( SRAM_OE0 = B"1", SRAM_OE1 = B"0");
   STWE0 : MACHINE OF BITS(/SRAM_WE0)
           WITH STATES ( SRAM_WE00 = B"1", SRAM_WE01 = B"0");
   STWE1 : MACHINE OF BITS(/SRAM_WE1)
           WITH STATES ( SRAM_WE10 = B"1", SRAM_WE11 = B"0");
   STWE2 : MACHINE OF BITS(/SRAM_WE2)
           WITH STATES ( SRAM_WE20 = B"1", SRAM_WE21 = B"0");
   STWE3 : MACHINE OF BITS(/SRAM_WE3)
           WITH STATES ( SRAM_WE30 = B"1", SRAM_WE31 = B"0");


    INT : MACHINE WITH STATES
                (
                INTS2 ,
                INTS3 ,
                INTS4 ,
                INTS5 ,
                INTS6
                );

    ARB : MACHINE WITH STATES
                                        (
                                        ARB0 ,
                                        ARB1 ,
                                        ARB2 ,
                                        ARB3 ,
```

```
                                        ARB4 ,
                                        ARB5 ,
                                        ARB6 ,
                                        ARB7 ,
                                        ARB8 ,
                                        ARB9 ,
                                        ARB10,
                                        ARB11
                                        );


BEGIN

% Inverted 2x Clock is used for SRAM Write State machine %

  CLK21 = !CLK2;

% PLX 9060 PCI GRANT SHOULD BE TRISTATED FROM FPGA WHEN
  CARD IS CONFIGURED FOR HOST ADAPTER %

 /9060_GNT_T = TRI(/9060_GNT, ARBSEL);

 % ADDRESS STROBE GENERATION FOR 401GF CYCLES %

 401_X1.clk = CLK1;
 401_X1.prn = /RESET;
 401_X1.clrn= /ALE;
 401_X1.d   = VCC;

--  !401_X1 = (!/ALE # (!401_X1 & !DCLK1)) & /RESET;

  /ADS    = TRI(401_X1, !401_HOLDA);
 Temp.clk = GLOBAL(CLK2);
 Temp.d = temp.q;

% SRAM lower address generation %
 SLA2.clk  =  CLK1;
 SLA2.clrn = !(!/RESET # (!/ADS & !LA2 & /RESET)) ;
 SLA2.prn  = !(!/ADS & LA2 & /RESET) ;
 SLA2.d    = ((!SLA2 & !/S_READY) # (SLA2 & /S_READY ));
 D_SLA2 = SLA2.q ;

 SLA3.clk  =  CLK1;
 SLA3.clrn = !(!/RESET # (!/ADS & !LA3 & /RESET)) ;
 SLA3.prn  = !(!/ADS & LA3 & /RESET) ;
 SLA3.d    = ((((!SLA3 & SLA2) # (SLA3 & !SLA2))&!/S_READY) #
             (/S_READY & SLA3));
 D_SLA3 = SLA3.q;
```

```
% SYNCHRONIZATION OF PUSH BUTTON CRITICAL INTERRUPT TO
  PROCESSOR CLOCK %
/PUSH_CRIT_NODE.clk = CLK1;
/PUSH_CRIT_NODE.PRN = /RESET;
/PUSH_CRIT_NODE.D = /PUSH_CRIT_INT;

INT_REMOVE_NODE.clk = CLK1;
INT_REMOVE_NODE.CLRN = /RESET;
INT_REMOVE_NODE.D = INT_REMOVE;

% Serial EEPROM read/write port Control Signals %
SerE2_OE = !/OE & !/SerE2_CS;
!/SerE2_WE   = !/WE & !/SerE2_CS;

% Delayed Write enable Generation %
/D_WE.clk = CLK1;
/D_WE.PRN = /RESET;
/D_WE.D = /WE;

% Ethernet Remote DMA Latch Control Signals %
% /BE0 is address A0 in 8 bit bus %

!/WACK    = !/401_RDY & W_/R & PRQ & !/LATCH_CS  & /BE0;
!/RACK    = !/401_RDY & !W_/R & PRQ & !/LATCH_CS  & /BE0;
!R_DMA_WR = PRQ & !/WE & !/LATCH_CS & /BE0;

% Direction  signal will be "LOW" 1. 401GF remote DMA read
                                  2. Ethernet register read %
!646_DIR  = ((PRQ & !/OE & !/LATCH_CS & /BE0)
            # (!/ENET_CS & !/OE & !/ACK));

% OUTPUT enable will be "LOW" 1. 401GF remote DMA read
                              2. Ethernet register read/write
                              3. 646 latched data read by ethernet
                                 controller %

!/646_OE = ((PRQ & !/OE & !/LATCH_CS & /BE0) # (!/ENET_CS & !/OE &
                          !/ACK)
           # (!/ENET_CS & !/ACK & (!/WE # !/D_WE)) # !/PRD) ;

% F646 read time data transfer data control signals %
!SAB     = !/ENET_CS & (!/WE # !/D_WE) & !/ACK;
!SBA     = !/ENET_CS & !/OE & !/ACK;
```

```
% 401GF CRITICAL INTERRUPT GENERATION FOR PUSHBUTTON & PCI SERR %
% Power ON critical interrupt is bypassed %

INT.clk = CLK1;
INT. reset = !/RESET;

CASE INT IS
   WHEN INTS2 =>
       /401_CRIT_INT = VCC;
       IF !/PUSH_CRIT_NODE THEN INT = INTS3; % Checking for push button
                           press %
       ELSIF  !/PCI_SERR   THEN INT = INTS5; % Checking for pci System
                           error %
       ELSE INT = INTS2;
       END IF;

   WHEN INTS3 =>
        /401_CRIT_INT = GND;
        IF INT_REMOVE_NODE THEN INT = INTS4; % Checking for interrupt
                           remove condition %
        ELSE INT =INTS3;
        END IF;
   WHEN INTS4 =>
        /401_CRIT_INT = VCC;
        IF /PUSH_CRIT_NODE THEN INT = INTS2; % Checking for Push button
                           release %
        ELSE INT = INTS4;
        END IF;

   WHEN INTS5 =>
        /401_CRIT_INT = GND;
        IF INT_REMOVE_NODE THEN INT = INTS6; % Checking for interrupt
                           remove condition %
        ELSE INT =INTS5;
        END IF;
   WHEN INTS6 =>
        /401_CRIT_INT = VCC;
        IF /PCI_SERR THEN INT = INTS2;
        ELSE INT = INTS6;
        END IF;

 END CASE;


% 401GF READY GENERATION %

!/401_RDY = TRI(!/S_READY, SR);
```

```
STSR.clk = CLK1;
STSR.reset = !/RESET;
STRDY.clk = CLK1;
STRDY.reset = !/RESET;
STSOE.clk =  CLK1;
STSOE.reset = !/RESET;
STWE0.clk = CLK21;
STWE0.reset=!/RESET;
STWE1.clk = CLK21;
STWE1.reset=!/RESET;
STWE2.clk = CLK21;
STWE2.reset=!/RESET;
STWE3.clk = CLK21;
STWE3.reset=!/RESET;

% To Distinguish Between Idle State and Cycle in Progress %

CASE STSR IS
  WHEN SR0 =>
     IF (!/ADS & !A30 & !A29 & A28 & !A27 & !A26 & !A25 & !A24 & !A23 &
                          !A22 & !A22 & !A21 & !A20) THEN
        STSR = SR1;
    ELSE STSR = SR0;
     END IF;

  WHEN SR1 =>
      IF !/BLAST THEN STSR = SR0;
      ELSE STSR = SR1;
      END IF;

END CASE;

% Ready signal generation for SRAM cycles %

CASE STRDY IS
  WHEN S_READY0 =>
     IF (!/ADS  & !A30 & !A29 & A28 & !A27 & !A26 & !A25 & !A24 & !A23
                          & !A22 & !A22 & !A21 & !A20) THEN STRDY =
                          S_READY1;
    ELSE STRDY = S_READY0;
     END IF;

  WHEN S_READY1 =>
      IF !/BLAST  THEN STRDY = S_READY0;
      ELSE STRDY = S_READY1;
      END IF;

END CASE;
```

```
% SRAM OUTPUT ENABLE %

CASE STSOE IS
  WHEN SRAM_OE0 =>
     IF (!/ADS & !W_/R & !A30 & !A29 & A28 & !A27 & !A26 & !A25 & !A24
                         & !A23 & !A22 & !A22 & !A21 & !A20) THEN
                         STSOE = SRAM_OE1;
    ELSE STSOE = SRAM_OE0;
     END IF;

  WHEN SRAM_OE1 =>
      IF !/BLAST THEN STSOE = SRAM_OE0;
      ELSE STSOE = SRAM_OE1;
      END IF;

END CASE;

% Least SRAM Write Byte Enable %

CASE STWE0 IS
  WHEN SRAM_WE00 =>
     IF (SR & DCLK1 & W_/R & !/BE0) THEN STWE0 = SRAM_WE01;
    ELSE STWE0 = SRAM_WE00;
     END IF;

  WHEN SRAM_WE01 =>
      IF (!/BLAST # !DCLK1) THEN STWE0 = SRAM_WE00;
      ELSE STWE0 = SRAM_WE01;
      END IF;

END CASE;


% SRAM WRITE BYTE ENABLE ONE %

CASE STWE1 IS
  WHEN SRAM_WE10 =>
     IF (SR & DCLK1 & W_/R & !/BE1) THEN STWE1 = SRAM_WE11;
    ELSE STWE1 = SRAM_WE10;
     END IF;
  WHEN SRAM_WE11 =>
      IF (!/BLAST # !DCLK1) THEN STWE1 = SRAM_WE10;
      ELSE STWE1 = SRAM_WE11;
      END IF;
END CASE;

% SRAM WRITE BYTE ENABLE2 %
```

```
CASE STWE2 IS
  WHEN SRAM_WE20 =>
     IF (SR & DCLK1 & W_/R & !/BE2) THEN STWE2 = SRAM_WE21;
    ELSE STWE2 = SRAM_WE20;
     END IF;
  WHEN SRAM_WE21 =>
      IF (!/BLAST # !DCLK1) THEN STWE2 = SRAM_WE20;
      ELSE STWE2 = SRAM_WE21;
      END IF;
END CASE;

% MOST SRAM WRITE BYTE ENABLE%
CASE STWE3 IS
  WHEN SRAM_WE30 =>
     IF (SR & DCLK1 & W_/R & !/BE3) THEN STWE3 = SRAM_WE31;
    ELSE STWE3 = SRAM_WE30;
     END IF;
  WHEN SRAM_WE31 =>
     IF (!/BLAST # !DCLK1) THEN STWE3 = SRAM_WE30;
     ELSE STWE3 = SRAM_WE31;
     END IF;
END CASE;


% ARBITRATION  STATE MACHINE %
ARB.clk = CLK1;
ARB.reset = !/RESET;

CASE ARB IS

WHEN ARB0 =>
    401_HOLD = GND;
    9060_HOLDA = GND;
    /SQBG = VCC;
    /PCI_GNT1 = VCC;
   /PCI_GNT2 = VCC;
    IF !/9060_REQ THEN /9060_GNT = GND;
    ELSE /9060_GNT = VCC;
    END IF;
    IF (9060_HOLD & !ARBSEL) THEN ARB = ARB1;
    ELSIF (!9060_HOLD & !/SQBR & !ARBSEL) THEN ARB = ARB4;
    ELSIF (9060_HOLD & ARBSEL ) THEN ARB = ARB1;
    ELSIF (!/9060_REQ & ARBSEL)  THEN ARB = ARB0;
   ELSIF (!/SQBR & /9060_REQ & ARBSEL) THEN ARB = ARB4;
    ELSIF (!/PCI_REQ1 & /SQBR & /9060_REQ & ARBSEL) THEN ARB = ARB7;
    ELSIF (!/PCI_REQ2 & /PCI_REQ1 & /SQBR & /9060_REQ & ARBSEL)  THEN
                          ARB = ARB10;
```

```
   ELSE ARB = ARB0;
   END IF;

WHEN ARB1 =>
   401_HOLD = VCC;
   9060_HOLDA = GND;
  IF !/9060_REQ THEN /9060_GNT = GND;
   ELSE /9060_GNT = VCC;
   END IF;
   /SQBG = VCC;
   /PCI_GNT1 = VCC;
  /PCI_GNT2 = VCC;
  IF 401_HOLDA THEN ARB = ARB2;
   ELSE ARB = ARB1;
   END IF;

WHEN ARB2 =>
   401_HOLD = VCC;
   9060_HOLDA = VCC;
  IF !/9060_REQ THEN /9060_GNT = GND;
   ELSE /9060_GNT = VCC;
   END IF;
   /SQBG = VCC;
   /PCI_GNT1 = VCC;
  /PCI_GNT2 = VCC;
  IF !9060_HOLD THEN ARB = ARB3;
   ELSE ARB = ARB2;
   END IF;

WHEN ARB3 =>
  401_HOLD = GND;
   9060_HOLDA = GND;
  IF !/9060_REQ THEN /9060_GNT = GND;
   ELSE /9060_GNT = VCC;
   END IF;
  /SQBG = VCC;
   /PCI_GNT1 = VCC;
  /PCI_GNT2 = VCC;
  IF !401_HOLDA THEN ARB = ARB0;
   ELSE ARB = ARB3;
   END IF;

WHEN ARB4 =>
   401_HOLD = VCC;
   9060_HOLDA = GND;
  IF !/9060_REQ THEN /9060_GNT = GND;
   ELSE /9060_GNT = VCC;
   END IF;
```

```
    /SQBG = VCC;
    /PCI_GNT1 = VCC;
   /PCI_GNT2 = VCC;
   IF 401_HOLDA THEN ARB = ARB5;
    ELSE ARB = ARB4;
    END IF;

 WHEN ARB5 =>
    401_HOLD = VCC;
    9060_HOLDA = GND;
   IF !/9060_REQ THEN /9060_GNT = GND;
    ELSE /9060_GNT = VCC;
    END IF;
    /SQBG = GND;
    /PCI_GNT1 = VCC;
   /PCI_GNT2 = VCC;
   IF /SQBR THEN ARB = ARB6;
    ELSE ARB = ARB5;
    END IF;

 WHEN ARB6 =>
   401_HOLD = GND;
    9060_HOLDA = GND;
   IF !/9060_REQ THEN /9060_GNT = GND;
    ELSE /9060_GNT = VCC;
    END IF;
   /SQBG = VCC;
    /PCI_GNT1 = VCC;
   /PCI_GNT2 = VCC;
   IF !401_HOLDA THEN ARB = ARB0;
    ELSE ARB = ARB6;
    END IF;

 WHEN ARB7 =>
    401_HOLD = VCC;
    9060_HOLDA = GND;
   /9060_GNT = VCC;
    /SQBG = VCC;
    /PCI_GNT1 = VCC;
   /PCI_GNT2 = VCC;
    IF 401_HOLDA THEN ARB = ARB8;
    ELSE ARB = ARB7;
    END IF;

 WHEN ARB8 =>
    401_HOLD = VCC;
    IF 9060_HOLD THEN 9060_HOLDA = VCC;
    ELSE 9060_HOLDA = GND;
```

```
    END IF;
   /9060_GNT = VCC;
    /SQBG = VCC;
    /PCI_GNT1 = GND;
   /PCI_GNT2 = VCC;
   IF ((BUSREQ # !/SQBR) & !/PCI_REQ1) THEN ARB = ARB9;
    ELSIF /PCI_REQ1 THEN ARB = ARB0;
    ELSE ARB = ARB8;
    END IF;

 WHEN ARB9 =>
    401_HOLD = VCC;
    IF 9060_HOLD THEN 9060_HOLDA = VCC;
    ELSE 9060_HOLDA = GND;
    END IF;
   /9060_GNT = VCC;
    /SQBG = VCC;
    /PCI_GNT1 = VCC;
   /PCI_GNT2 = VCC;
   IF (!/FRAME # !/IRDY # 9060_HOLD) THEN ARB = ARB9;
    ELSE ARB = ARB0;
    END IF;
%
  ARBITRATION LOGIC FOR SECOND PCI SLOT
%
 WHEN ARB10 =>
    401_HOLD = VCC;
    9060_HOLDA = GND;
   /9060_GNT = VCC;
    /SQBG = VCC;
    /PCI_GNT1 = VCC;
   /PCI_GNT2 = VCC;
    IF 401_HOLDA THEN ARB = ARB11;
    ELSE ARB = ARB10;
    END IF;

 WHEN ARB11 =>
   401_HOLD = VCC;
    IF 9060_HOLD THEN 9060_HOLDA = VCC;
    ELSE 9060_HOLDA = GND;
    END IF;
   /9060_GNT = VCC;
   /SQBG = VCC;
    /PCI_GNT1 = VCC;
   /PCI_GNT2 = GND;
   IF ((BUSREQ # !/SQBR )&!/PCI_REQ2) THEN ARB = ARB9;
    ELSIF (!/PCI_REQ1 & !/PCI_REQ2) THEN ARB =ARB7;
    ELSIF /PCI_REQ2 THEN ARB = ARB0;
```

```
    ELSE ARB = ARB11;
    END IF;

END CASE;

END;
```

D

401 EVB Bill of Materials

Table D-1. 401 EVB Bill of Materials

Part Name	Part #	Ref Des	Qty	Manufac Name	Manufac Part #	JEDEC_Type	Part Description
29F040ROM_MECH_MECH-BASE	82G6496	M2,M7	2	AMD	AM29F040-120JC	PART	512KX8 FLASH ROM
401_SMD-BASE	XXXXXXX	U1	1	IBM	IBM25-PPC401	QFP80_5MMSKT	401 50MHZ
74ALS38_SMD-BASE	68X2725	U32	1	TI	SN74ALS38BDR	SO14	OPEN COLLECTOR NAND
74ALS646_SMD-BASE	6448368	U43	1	NATIONAL	DMALS646WMX	SO24W	OCTAL BUS TRANS W/REGISTER
74F02_SMD-BASE	6480477	U69	1	NATIONAL	74F02SCX	SO14	DUAL 2-INPUT NOR GATE
74F04_SMD-BASE	61X9235	U9, U39	2	NATL	FO4	SO14	HEX INVERTER
74F126_SMD-BASE	09F1697	U28	1	PHILLIPS	N74F126D-T	SO14	QUAD BUFFER WITH 3-STATE OUTPUT
74F244_SMD-BASE	6480438	U35	1	PHILIPS	74F244	SO20W	OCTAL BUFF/DRIVER
74F257_SMD-BASE	72X8199	U11-U14, U19-U22	8	NATIONAL	N74F257AD-T	SO16	QUAD 2-1 LINE MUX
74F273_SMD-BASE	72X8346	U23	1	PHILLIPS	N74F273D-T	SO20	OCTAL D FLIP-FLOP
82C59_SMD-BASE	23F0299	U5, U33	2	HARRIS	CS82C59A-12	PLCC28	PROGRAMMABLE INTERRUPT CTRL
ABT04_SMD-BASE	XXXXXXX	U4	1	PHILLIPS	74ABT04D	SO14	NOT GATE
ABT16373_SMD-BASE	49G3313	U15, U16	2	TI	SN74ABT16373ADLR	SSOP48	16-BIT TRANSPARENT D-LATCHES W/3-ST OUT
ABT241_SMD-BASE	XXXXXXX	U29-U31	3	TI	SN74ABT241	SO20	74ABT241 BUFFER 3 STATES

Table D-1. 401 EVB Bill of Materials (Continued)

Part Name	Part #	Ref Des	Qty	Manufac Name	Manufac Part #	JEDEC_Type	Part Description
ABT245_SMD-BASE	40G7769	U17	1	TI	SN74ABT245BDWR	SO20	74ABT245 TRAN-CEIVER
ABT573_SMD-BASE	49G3254	U49	1	TI	SN74ABT573DWR	SO20	74ABT573 TRANSPAR-ENT LATCH 3-STATE
APEM_DIP-BASE	XXXXXXX	SW1, SW2	2	APEM	A2216	SW_APEM-A2216	MOMENTARY SWITCH
BERG1X2_DIP-BASE	6181127	J10, J20-J22, J26, J39-J42, TP1-TP4	13	BERG	69190-502	BERG1X2	1X2 100MIL HEADER VERTICAL
BERG1X3_DIP-BASE	1501831	J9, J12, J13, J27-J30, J32, J43	9	MOLEX	90368-2703	BERG1X3	1X3 100MIL HEADER VERTICAL
BUSHEADER_DIP-BASE	XXXXXXX	J15, J16	2	MOLEX	15-44-3216	CONN_2X16_MLX15-44-3216	2X16 100MIL FEMALE HEADER
CAPACITOR-0.001UF, 20%	69G2918	C93, C146, C147	3	AVX	08055C102MAT2A	SMC0805	CAPACITOR
CAPACITOR-0.01UF,20%	41F0313	C1-C4, C9, C17, C19, C21, C22, C24-C 29, C34, C35, C42, C44, C54-C 57, C59, C60, C69, C75, C85, C98, C107-C109, C114-C117, C120- C123, C132, C134, C136, C137, C140, C141, C144, C152, C158, C163, C165, C194, C196, C203, C204, C213, C221, C222, C237, C250- C256	66	KEMET	C0805C103M5RAC	SMC0805	CAPACITOR

Table D-1. 401 EVB Bill of Materials (Continued)

Part Name	Part #	Ref Des	Qty	Manufac Name	Manufac Part #	JEDEC_Type	Part Description
CAPACITOR-0.1UF,20%	78G9541	C5-C7, C10, C11, C13, C16, C18, C20, C23, C30, C40, C41, C45, C48, C49, C52, C53, C58, C77, C78, C84, C88-C90, C92, C94, C95, C97, C100-C102, C104, C106, C113, C124-C131, C133, C135, C138, C139, C142, C143, C145, C151, C154, C159, C166-C168, C172, C178-C191, C193, C197-C201, C205, C207-C209, C212, C226, C229, C233-C236, C241-C248, C257, C258, C260-C262	101	AVX	08053E104MAT2A	SMC0805	CAPACITOR
CAPACITOR-10PF,5%	68X6212	C38, C73	2	KYOCERA	08055A100JAT2A	SMC0805	CAPACITOR
CAPACITOR-2.2NF,10%	57G8771	C214, C219, C220, C240	4	AVX	08055C222KAT2A	SMC0805	CAPACITOR
CAPACITOR-220PF,5%	03G9606	C61-C63, C65, C66, C103, C105, C110, C118, C119, C211, C215-C218, C232	16	KYOCERA	08055A221JAT2A	SMC0805	CAPACITOR

Table D-1. 401 EVB Bill of Materials (Continued)

Part Name	Part #	Ref Des	Qty	Manufac Name	Manufac Part #	JEDEC_Type	Part Description
CAPACITOR-470PF,5%	41F0311	C148, C149	2	AVX	08055A471JAT2A	SMC0805	CAPACITOR
CAPACITOR-DO_NOT_POP, 20%	-NONE--	C153, C155, C160-C162, C164, C169, C224, C225, C239	10			SMC0805	CAPACITOR
CONNBNC_DIP-BASE	XXXXXXX	J23	1	AMP	227161-9	CONN_BNC_AMP22 7677-1	BNC RECTANGULAR JACK
CONNPARAL_DIP-BASE	90X8089	J5	1	AMP	747846-4	CONN_PIH25D_SH	25 PTH HEADER ELL
CONNPCI_DIP-BASE	72G0316	J8	1	AMP	646255-1	PCI2X605V	2X60 32 BIT PCI CONNECTOR
CONNPOWER_DIP-BASE	55X8085	J11	1	MOLEX	15-48-0212	CONN_55X8085	1X12 0 156 CL FRET LCK HDR
CONNRISC_DIP-BASE	XXXXXXX	J17	1	THOMAS & BETTS	PS-16DS-HBA	CONNESP403	2X8 100MIL KEY HEADER VERTICAL Note: remove pin 14 before assembly
CONNS72_DIP-BASE	64F5806	J1,J2	2	AMP	821997-3	CONN_64F5806	72 PIN SIMM CONNECTOR
CONNSERIAL_DIP-BASE	90X8092	J6, J19	2	AMP	748879-1	CONN_90X8092	9 PTH D-SHELL HEADER
CONNSQUALL_DIP-BASE	XXXXXXX	J4	1	AMP	1-104652-0	CONN_SMC_AMP1-104652-0	2X50 SQUALL CONNECTOR
CRYSTL-DO_NOT_POP	-NONE-	Y1	1	NONE	NONE	XTALSMT2	CRYSTAL
DC_CONV_DIP-BASE	XXXXXXX	U59	1	VALOR	PM7202	DC-DCCNVT_PM72	ETHERNET DC CONVERTER

Table D-1. 401 EVB Bill of Materials (Continued)

Part Name	Part #	Ref Des	Qty	Manufac Name	Manufac Part #	JEDEC_Type	Part Description
DP83902A_SMD-BASE	59G7143	U38	1	NATIONAL	DP83902AVJG	100QFP_S	STNIC ETHERNET CNTL
DP8392C_SMD-BASE	XXXXXXX	U58	1	NATIONAL	DP8392CV	PLCC28	COAX TRANCEIVER INTERFACE
DS1233_SMD-BASE	05H1621	U6	1	DALLAS	DS1233DZ-10	SOT223	VOLTAGE MONITOR WITH RESET
DS1643L_SKT_SMD-BASE	XXXXXXX	J14	1	MCKENZIE	PLCC26P-SMT-3	PLCC26SKT	RTC SOCKET
DS1643L_SMD-BASE	XXXXXXX	M8	1	DALLAS	DS1643L-120	PART	NVRAM RTC
ELCAP-10UF,20%	71F7911	C79, C238	2	NEC	TESVEC1C106M12R	SMC2312	CAPACITOR
ELCAP-22UF,20%	58F1742	C91, C96	2	KEMET	T496D226M016AS	CAP33UF3SMT	CAPACITOR
ELCAP-33UF,20%	57G9281	C8, C14, C31-C33, C36, C37, C39, C43, C46, C47, C50, C51, C64, C67, C68, C70, C71, C74, C76, C80-C82, C86, C111, C112, C150, C170, C171, C174-C177, C192, C195, C202, C230	37	KEMET	T491D336M016AS	CAP33UF3SMT	CAPACITOR
EPM7128_SMD-BASE	XXXXXXX	M3-M5	3	ALTERA	EPM7128EQC100-7	PART	FPGA
F_BEAD-MID-Z	26F4864	FB1-FB3, FB5	4	TDK	HF50ACB-322513-T	SMC1210	FERRITE BEAD
GURU_SMD-BASE	-NONE-	U34	1			100 QFP	SOCKET (NOT USED, EPM7128_SMD-BASE SOLDERED DIRECTLY)

Table D-1. 401 EVB Bill of Materials (Continued)

Part Name	Part #	Ref Des	Qty	Manufac Name	Manufac Part #	JEDEC_Type	Part Description
HM6264_SMD-BASE	72X7415	U51	1	HITACHI	HM6264LFP-10	SO28XW	SRAM8KX8
LCD162B_DIP-BASE	XXXXXXX	M6	1	FEMA	CM162B-SGT1LY-R10-HR	PART	LCD
LCDSKT_DIP-BASE	XXXXXXX	J31	1		15-44-3207	FEMA162BSKT	2X7 BERG FEMALE SOCKET
LED_A-GREEN	42G3090	CR1, CR3, CR4	3	HP	HLMP-1790-002	LED_100	XX
LED_A-YELLOW	42G3089	CR2	1	HP	HLMP-1719-002	LED_100	XX
LT1085_DIP-BASE	05H1511	U2	1	LINEART-ECH	LT1085CT-3.3	TO220V_H-S_AVD	VOLT REGULATOR
LVC16244_SMD-BASE	40H8824	U7, U8, U10, U18	4	TI	SN74LVCH16244A	TSSOP48	16-BIT BUFFER TRI-STATE
MAVEN_SMD-BASE	-NONE-	U24	1			100 QFP	SOCKET (NOT USED, EPM7128_SMD-BASE SOLDERED DIRECTLY)
MC88915_SMD-BASE	XXXXXXX	U3	1	MOTOROLA	MC88915FN70	PLCC28	CLOCK CHIP
OSCLR-20.0MHZ	42G3137	Y3	1	EPSON	SG615P-20.00000MHZ-100PPM	OSCSMT4	
OSCLR-24.0MHZ	87F5265	Y4	1	EPSON	SG615P-24.0MHZ-8NS	OSCSMT4	
PAL_20-DO_NOT_POP	XXXXXXX	J44	1	XXX	XXXXXXXX	PLCC20	
PC16553_SMD-BASE	34G2684	U48	1	NATIONAL	PC16553DV	PLCC68	UART PARALLEL CNTL
PCI9060_SMD-BASE	XXXXXXX	U52	1	PLX TECH	PCI9060	QFP208_5MM	PCI BUS INTERFACE

Table D-1. 401 EVB Bill of Materials (Continued)

Part Name	Part #	Ref Des	Qty	Manufac Name	Manufac Part #	JEDEC_Type	Part Description
PLCC32SKT_SMD-BASE	10G7624	J3, J18	2	AMP	821977-1	PLCC32SKT	32 PIN PLCC SOCKET
RESISTOR-0,5%	98F1665	R4, R168, R275	3	ROHM	MCR10EZHM0R00	SMC0805	RESISTOR
RESISTOR-1.0M,5%	98F1667	R135	1	PANASONIC	ERJ6GVYJ105S	SMC0805	RESISTOR
RESISTOR-1.2M,5%	03G9683	R269	1	PANASONIC	ERJ6GVYJ125S	SMC0805	RESISTOR
RESISTOR-1.5K,5%	98F1741	R49-R51, R53, R266	5	ROHM	MCR10EZHMJW152	SMC0805	RESISTOR
RESISTOR-100,5%	41F0328	R8, R13, R22, R48, R52, R80, R131, R154, R156, R159, R160, R162, R514	13	PANASONIC	ERJ6GYVJ101V	SMC0805	RESISTOR

Table D-1. 401 EVB Bill of Materials (Continued)

Part Name	Part #	Ref Des	Qty	Manufac Name	Manufac Part #	JEDEC_Type	Part Description
RESISTOR-10K,5%	41F0337	R3, R9, R28, R29, R46, R55, R59-R64, R66-R73, R75, R76, R79, R84-R95, R107-R109, R111, R113, R115, R116, R118, R119, R126-R130, R132, R133, R136, R137, R147, R151-R153, R157, R158, R161, R166, R170, R176, R177, R200, R214, R223, R226, R280-R282, R284-R291, R2803	80	PANASONIC	ERJ6GEYJ103V	SMC0805	RESISTOR
RESISTOR-1K,1%	31F1911	R42	1	PANASONIC	ERJ6VNF1001S	SMC0805	RESISTOR
RESISTOR-1K,5%	41F0333	R34-R41, R82, R103, R104, R117, R134	13	PANASONIC	ERJ6GEYJ102V	SMC0805	RESISTOR

Table D-1. 401 EVB Bill of Materials (Continued)

Part Name	Part #	Ref Des	Qty	Manufac Name	Manufac Part #	JEDEC_Type	Part Description
RESISTOR-22,5%	98F1736	R2, R6, R10-R12, R14, R15, R17-R21, R23-R27, R54, R56, R65, R121, R122, R155, R163, R164, R169, R224, R225, R229, R231-R233, R237, R242, R243, R245, R246, R249-R251	40	PANASONIC	ERJ6GEYJ220V	SMC0805	RESISTOR
RESISTOR-274,1%	40G7234	R143, R144	2	PANASONIC	ERJ6ENF2740V	SMC0805	RESISTOR
RESISTOR-300,5%	98F1674	R47, R138	2	ROHM	MCR10EZHMJW301	SMC0805	RESISTOR
RESISTOR-330,5%	08G4750	R105, R148-R150,R165	5	ROHM	MCR10EZHMJW331	SMC0805	RESISTOR
RESISTOR-3.9K,5%	42G3067	R175	1	PANASONIC	ERJ6GEYJ392V	SMC0805	RESISTOR
RESISTOR-4.7K,5%	41F0336	R30-R33, R43-R45, R77, R78, R81, R96-R102, R106, R110, R146	20	PANASONIC	ERJ6GEYJ472V	SMC0805	RESISTOR
RESISTOR-47,5%	03G9708	R16, R58	2	ROHM	MCR10EZHLJW470	SMC0805	RESISTOR
RESISTOR-470K,5%	03G9709	R57	1	ROHM	MCR10EZHLJW474	SMC0805	RESISTOR
RESISTOR-49.9,5%	40G6869	R141, R142	2	PANASONIC	ERJ6ENF49R9V	SMC0805	RESISTOR
RESISTOR-66.5,1%	40G6881	R139, R140	2	PANASONIC	ERJ6ENF66R5V	SMC0805	RESISTOR
RESISTOR-75,5%	82G6645	R123, R124	2	PANASONIC	ERJ6GEYJ750V	SMC0805	RESISTOR

Table D-1. 401 EVB Bill of Materials (Continued)

Part Name	Part #	Ref Des	Qty	Manufac Name	Manufac Part #	JEDEC_Type	Part Description
RESISTOR-806,1%	40G6968	R145	1	PANASONIC	ERJ6ENF8060V	SMC0805	RESISTOR
RESISTOR-DO_NOT_POP, 5%	-NONE--	R00, R01, R1, R5, R83, R120	6			SMC0805	RESISTOR
RJ45_DIP-BASE	59G2489	J24	1	AMP	555164-1	CONN_RJ45	RJ45 CONN
SAGE_SMD-BASE	-NONE-	U54	1			100 QFP	SOCKET (NOT USED, EPM7128_SMD-BASE SOLDERED DIRECTLY)
SCREW_BOARD	XXXXXXX		6	NAPPCO FASTENER			NYLON SCREW, BINDER HEAD, 3/8" X 4-40
SCREW_LCD	XXXXXXX		4	NAPPCO FASTENER			NYLON SCREW, BINDER HEAD, 1/4" X 2-56
SLIPSINK_DIP-BASE	XXXXXXX	M1	1	AAVID	576802B00000	PART	HEATSINK VOLTREG
SN75LV4737A_SMD-BASE	40H9046	U36, U65	2	TI	SN75LV4737A	SSOP28	RS232 TRANSCEIVER
SRAM128KX8_SKT_SMD-BASE	XXXXXXX	J35-J38	4	AMP	822362-1	SOJ32W_400SKT	SRAM128KX8 SOCKET
SRAM128KX8_SMD-BASE	XXXXXXX	M11-M14	4	MOTOROLA	MCM6726BWJ-12	PART	SRAM128KX8
STANDOFF_BOARD	XXXXXXX		6	NAPPCO FASTENER			NYLON STANDOFF, 1/2" TALL, 3/16" DIA, 4-40 THREAD
STANDOFF_LCD	XXXXXXX		2	NAPPCO FASTENER			NYLON STANDOFF, 7/16" TALL, 3/16" DIA, 2-56 THREAD
XFORM_COAX_DIP-BASE	XXXXXXX	U50	1	VALOR	LT6032	DIP16_300	ETHERNET TRANS-FORMER

Table D-1. 401 EVB Bill of Materials (Continued)

Part Name	Part #	Ref Des	Qty	Manufac Name	Manufac Part #	JEDEC_Type	Part Description
XFORM_TWST_DIP-BASE	XXXXXXX	U37	1	VALOR	FL1012	FILTER_VALOR-FL1012	ETHERNET TRANS-FORMER
XTAL-25.00000MHZ	40H8794	Y2	1	DAISHINKU	HC49U-25MHZ-30/50PPM-18PF	CLP_XTAL	

Index

A

Alignment Exception Support Library 9-1
ANSI C I/O Library 9-1
ANSI C Library 9-1
ANSI C Math Library 9-1
async safe 10-1
async_init() function 10-10
asyncLib.a library 9-4

B

biosenet_attach() function 10-11
Block Buffer Library 9-1
book
  conventions used xxi
    highlighting xxi
    numeric xxi
    syntax diagrams xxii
  how organized xx
  who should use this book xix
Boot Library 9-1, 9-4
booting the processor 5-28

C

C++ runtime support library 9-2
cancel safe 10-2
Clock Support Library 9-2
clock_set() function 10-13
clockchip_nvram_read() function 10-16, 10-21
clockchip_nvram_write() function 10-17, 10-22
clockchip_set() function 10-14, 10-15, 10-18, 10-19, 10-20
clockchip_set_calibration() function 10-24
clockchip_start() function 10-25
clockchip_stop() function 10-26
clockLib.a library 9-5
clockLib_init() function 10-27
connecting the EVB hardware 5-21
connectors
  ethernet 5-4
  expansion interface 5-8
  power 5-14
  RISCTrace 5-7
  RISCWatch 400 JTAG 5-7
  serial port 5-3
conventions used xxi
  highlighting xxi
  numeric xxi
  syntax diagrams xxii

D

dbg_ioLib_init() function 10-28
dcache_flush() function 10-29
dcache_invalidate() function 10-30
Debug Support Library 9-2
Device and File Support Library 9-2
device drivers
  asynchronous 9-7
  Ethernet 9-13
DOS File System Support Library 9-2
driver_install
  async_init 9-7
Dynamic Loader Library 9-2

E

enet_disable_ipinput() function 10-31
enet_enable_ipinput() function 10-32
enet_native_attach() function 10-33
enet_recv_packet() function 10-35
enet_send_packet() function 10-36
enetLib.a 9-5
Ethernet 9-13
Ethernet Device Driver Installation 9-13
ext_int_disable() function 10-37
ext_int_enable() function 10-38
ext_int_install() function 10-39
ext_int_query() function 10-40

F

Federal Communications Commission (FCC) Statement xxiii
File Transfer Protocol Support Library 9-2
Flash update utility 7-30
Floating Point Emulation Library 9-2
Floating Point Library 9-2
fpemul_init() function 10-41
functions
  async_init() 10-10
  biosenet_attach() 10-11
  clock_set() 10-13

clockchip_get() 10-14, 10-19
clockchip_get_calibration() 10-15, 10-20
clockchip_nvram_read() 10-16, 10-21
clockchip_nvram_write() 10-17, 10-22
clockchip_set() 10-14, 10-15, 10-18, 10-19, 10-20, 10-27
clockchip_set_calibration() 10-24
clockchip_start() 10-25
clockchip_stop() 10-26
clockLib_init() 10-27
dbg_ioLib_init() 10-28
dcache_flush() 10-29
dcache_invalidate() 10-30
enet_disable_ipinput() 10-31
enet_enable_ipinput() 10-32
enet_native_attach() 10-33
enet_recv_packet() 10-35
enet_send_packet() 10-36
ext_int_disable() 10-37
ext_int_enable() 10-38
ext_int_install() 10-39
ext_int_query() 10-40
fpemul_init() 10-41
ioLib_init() 10-42
lcd_init() 10-43
ppcAbend() 10-44
ppcAndMsr() 10-45
ppcCntlzw() 10-46
ppcDcbf() 10-47
ppcDcbi() 10-48
ppcDcbst() 10-49
ppcDcbz() 10-50
ppcDflush() 10-51
ppcEieio() 10-52
ppcHalt() 10-53
ppcIcbi() 10-54
ppclsync() 10-55
ppcMfbear() 10-56
ppcMfbesr0() 10-57
ppcMfbrcr0() - ppcMfbrcr7() 10-58
ppcMfcdbcr() 10-59
ppcMfdbcr() 10-60
ppcMfdbsr() 10-61
ppcMfdccr() 10-62
ppcMfdcwr() 10-63
ppcMfdear() 10-64
ppcMfesr() 10-65
ppcMfevpr() 10-66
ppcMfgpr1() 10-67
ppcMfgpr2() 10-68
ppcMfiac1() 10-69
ppcMficcr() 10-70
ppcMficdbdr() 10-71
ppcMfiocr() 10-72
ppcMfmsr() 10-73
ppcMfpit() 10-74
ppcMfpmcr0() 10-75
ppcMfpvr() 10-76
ppcMfsgr() 10-77
ppcMfsler() 10-78
ppcMfsprg1() • ppcMfsprg3() 10-79
ppcMfsprg1() • ppcMtsprg3() 10-103
ppcMfsrr0() 10-80
ppcMfsrr1() 10-81
ppcMfsrr2() 10-82
ppcMfsrr3() 10-83
ppcMftb() 10-84
ppcMftsr() 10-85
ppcMtbrcr0() - ppcMtbrcr7() 10-86
ppcMtcdbcr() 10-87
ppcMtdbcr() 10-89
ppcMtdbsr() 10-90
ppcMtdccr() 10-91
ppcMtdcwr() 10-92
ppcMtesr() 10-93
ppcMtevpr() 10-94
ppcMtiac1() 10-95
ppcMticcr() 10-96
ppcMtiocr() 10-97
ppcMtmsr() 10-98
ppcMtpit() 10-99
ppcMtpmcr0() 10-100
ppcMtsgr() 10-101
ppcMtsler() 10-102
ppcMtsrr1() 10-105
ppcMtsrr2() 10-106
ppcMtsrr3() 10-107
ppcMttb() 10-108
ppcMttcr() 10-109
ppcMttsr() 10-110

ppcOrMsr() 10-111
ppcSync() 10-112
ppMtdac1() 10-88
processor_speed() 10-113
s1dbprinf() 10-114
s2dbprinf() 10-115
timertick_install() 10-116
timertick_remove() 10-117
vs1dbprinf() 10-118

H

hardware components 1-1
cables and power supply 1-1
evaluation board 1-1
host system requirements
PC 2-2
RS/6000 2-1
Sun 2-3

I

I/O control 9-10, 9-15
IBM Embedded Systems Solution Center xxii
initialization 9-17
board bootstrap 9-17
Input/output Support Library 9-2
installing
ioLib.a library 9-4
ioLib_init() function 10-42

J

jumpers
setting 5-17

K

Kernel Abstract Data Types Library 9-2

L

LCD Library 9-2
lcd_init() function 10-43
library description
asyncLib.a 9-4
clockLib.a 9-5
enetLib.a 9-5
ioLib.a 9-4
ppcLib.a 9-5
rtx.o 9-3
rtxLib.a 9-3
tickLib.a 9-6

N

Network Support Library 9-2
NFS Support Library 9-2

O

opening asynchronous communication ports 9-9
OpenShell 9-2
OS Open kernel extensions 9-3
OS Open minimal kernel 9-3

P

PC host configuration 4-7
ethernet setup 4-14
ethernet setup for Windows 3.1 4-10
ethernet setup for Windows 95 4-12
ethernet setup for Windows NT 3.51 4-13
serial port setup 4-8
PC installation 3-4
RISCWatch debugger 3-7
software support package 3-4
PCMCIA ATA/IDE 9-2
PCMCIA card services/enabler 9-2
PCMCIA socket sevices 9-3
polled asynchronous I/O 9-12
PowerPC Low Level Access Support Library 9-2
PowerPC Low-Level Processor Access Support Library 9-5
ppcAbend() function 10-44
ppcAndMsr() function 10-45
ppcCntlzw() function 10-46
ppcDcbf() function 10-47
ppcDcbi() function 10-48
ppcDcbst() function 10-49
ppcDcbz() function 10-50
ppcDflusfh() function 10-51
ppcEieio() function 10-52
ppcHalt() function 10-53
ppcIcbi() function 10-54
ppcLib.a library 9-5
ppclsync() function 10-55
ppcMfbear() function 10-56

ppcMfbesr0() function 10-57
ppcMfbrcr0() - ppcMfbrcr7() functions 10-58
ppcMfcdbcr() function 10-59
ppcMfdbcr() function 10-60
ppcMfdbsr() function 10-61
ppcMfdccr() function 10-62
ppcMfdcwr() function 10-63
ppcMfdear() function 10-64
ppcMfesr() function 10-65
ppcMfevpr() function 10-66
ppcMfgpr1() function 10-67
ppcMfgpr2() function 10-68
ppcMfiac1() function 10-69
ppcMficcr() function 10-70
ppcMficdbdr() function 10-71
ppcMfiocr 10-72
ppcMfiocr() function 10-72
ppcMfmsr() function 10-73
ppcMfpit() function 10-74
ppcMfpmcr0() function 10-75
ppcMfpvr() function 10-76
ppcMfsgr() function 10-77
ppcMfsler() function 10-78
ppcMfsprg0() • ppcMfsprg3() function 10-79
ppcMfsrr0() function 10-80
ppcMfsrr1() function 10-81
ppcMfsrr2() function 10-82
ppcMfsrr3() function 10-83
ppcMftb() function 10-84
ppcMftsr() function 10-85
ppcMtbrcr0() - ppcMtbrcr7() functions 10-86
ppcMtcdbcr() function 10-87
ppcMtdac1() function 10-88
ppcMtdbcrar() function 10-89
ppcMtdbcrsr() function 10-90
ppcMtdccr() function 10-91
ppcMtdcwr() function 10-92
ppcMtdsisr() function 10-94
ppcMtesr() function 10-93
ppcMtiac1() function 10-95
ppcMticcr() function 10-96
ppcMtiocr() function 10-97
ppcMtmsr() function 10-98
ppcMtpit() function 10-99
ppcMtpmcr0() function 10-100
ppcMtsgr() function 10-101
ppcMtsler() function 10-102
ppcMtsprg0() • ppcMtsprg3() function 10-103
ppcMtsrr1() function 10-105
ppcMtsrr2() function 10-106
ppcMtsrr3() function 10-107
ppcMttb() function 10-108
ppcMttcr() function 10-109
ppcMttsr() function 10-110
ppcOrMsr() function 10-111
ppcSync() function 10-112
processor_speed() function 10-113
ptrace
  definitions A-4
    RD_ATTACH A-5
    RD_CONTINUE A-6
    RD_DETACH A-7
    RD_FILL A-8
    RD_KILL A-9
    RD_LDINFO A-10
    RD_LOAD A-12
    RD_LOGIN A-13
    RD_LOGOFF A-14
    RD_READ_D A-15
    RD_READ_DCR A-16
    RD_READ_GPR A-17
    RD_READ_GPR_MULT A-18
    RD_READ_I A-19
    RD_READ_I_MULT A-20
    RD_READ_SPR A-21
    RD_STATUS A-22
    RD_STOP_APPL A-23
    RD_WAIT A-24
    RD_WRITE_BLOCK A-25
    RD_WRITE_D A-26
    RD_WRITE_DCR A-27
    RD_WRITE_GPR A-28
    RD_WRITE_I A-29
    RD_WRITE_SPR A-30
    RL_LDINFO A-31
    RL_LOAD_REQ A-32
  MSGDATA structure A-1
  overview A-1

Q

Queue Library 9-2

R

RAM Disk Library 9-2
Rate Monotonic Scheduling (RMS) Library 9-2
Real_time Executive 9-3
Real-time Clock Interface Support Library 9-5
related publications xxiii
Remote Source Level Debug Library 9-2
Ring Buffer Library 9-2
ROM Boot Library 9-5
ROM monitor
  accessing 7-8
  bootp and tftp configuration for loads 7-2
    PC 7-4, 7-6
    RS/6000 7-2
  communication features 7-2
  menus 7-9
    changing IP addresses 7-15
    disabling the automatic display 7-21
    displaying the current configuration 7-22
    entering the debugger 7-19
    exiting the main menu 7-28
    initial ROM monitor menu 7-9
    saving the current configuration 7-23
    selecting boot devices 7-13
    selecting power-on tests 7-11
    using the ping test 7-17
  source code 7-1
  user functions 7-30
ROM Monitor Ethernet Interface Library 9-2
ROM monitor load format
  boot header B-3
  overview B-1
  section types B-1
    data section B-3
    first section B-2
    symbol section B-3
  sections types
    text section B-3
RPC Support Library 9-2
RS/6000 host configuration 4-1
  ethernet setup 4-5
  serial port setup 4-1
RS/6000 installation 3-1
  RISCWatch debugger 3-4
  software support package 3-1
rtx.o library 9-3
rtxLib.a library 9-3
Runtime Library 9-2

S

s1dbprintf() function 10-114
s2dbprintf() function 10-115
sample applications
  overview 8-1
  resolving problems 8-9
    bootp and tftp servers 8-10
    using the ping test 8-9
  ROM monitor flash image 8-2
  using 8-4
    Dhrystone benchmark 8-6
    timesamp program 8-8
    usr_samp program 8-7
SCSI Support Library 9-3
Serial Port Support Library 9-4
Serial Support Library 9-3
software support package 1-2
  application libraries and tools 1-3
  Dhrystone benchmark 1-3
  High C/C++ compiler 1-2
  RISCWatch 400 debugger 1-2
  ROM monitor 1-2
Software Timer Tick Support Library 9-6
Sun host configuration 4-13
Sun installation
  RISCWatch debugger 3-10
  software support package 3-7
Symbol Support Library 9-3

T

TCP/IP Protocol Support Library 9-3
Telnet Client Support Library 9-3
Telnet Daemon Support Library 9-3
tickLib.a library 9-6
Timer Tick Support 9-3
timertick_install() function 10-116
timertick_remove() function 10-117
tools 9-19
  eimgbld 9-23

elf2rom 9-19
hbranch 9-21
Trivial File Transfer Protocol Library 9-3
TTY Support Library 9-3

U

using a terminal emulator 5-25
PC terminal emulation 5-26
RS/6000 terminal emulation 5-26
Sun terminal emulation 5-27

V

vs1dbprintf() function 10-118

W

writing calls on asynchronous ports 9-9, 9-15